Oasys
LS-DYNA Environment

Oasys Ltd LS-DYNA Environment Software JavaScript API Manual

Version 23.0

For help and support from Oasys Ltd please contact:

UK

Tel: +44 121 213 3399
Email: dyna.support@arup.com

China

Tel: +86 21 3118 8875
Email: china.support@arup.com

India

Tel: +91 40 69019723 / 98
Email: india.support@arup.com

USA West

Tel: +1 415 940 0959
Email: us.support@arup.com

Web:www.oasys-software.com/dyna

or contact your local distributor.

LS-DYNA, LS-OPT and LS-PrePost are registered trademarks of Ansys, Inc.

PRIMER **PRIMER 1**
global class PRIMER 1
Keywords PRIMER 18
Airbag PRIMER 18
Boundary PRIMER 92
Comment PRIMER 210
Constrained PRIMER 227
Contact PRIMER 558
Control PRIMER 618
Damping PRIMER 681
Database PRIMER 790
Define PRIMER 883
DeformableToRigid PRIMER 1113
Element PRIMER 1141
Frequency Domain PRIMER 1575
Hourglass PRIMER 1642
IGA PRIMER 1661
Include PRIMER 2168
Initial PRIMER 2189
Integration PRIMER 2391
Interface PRIMER 2431
Load PRIMER 2486
Material PRIMER 2642
Node PRIMER 2678
Parameter PRIMER 2706
Part PRIMER 2717
Rigidwall PRIMER 2750
Section PRIMER 2778
Sensor PRIMER 2812
Set PRIMER 2877
Termination PRIMER 2908
Attached class PRIMER 2926
Belt class PRIMER 2930
Check class PRIMER 2961
Colour class PRIMER 2970
Conx class PRIMER 2973
Dummy class PRIMER 3014
File class PRIMER 3044
GeometrySurface class PRIMER 3064
Graphics class PRIMER 3088
Group class PRIMER 3097
Image class PRIMER 3127
Link class PRIMER 3130
Mechanism class PRIMER 3136
Model class PRIMER 3167
MorphBox class PRIMER 3195
MorphFlow class PRIMER 3222
MorphPoint class PRIMER 3247
Options class PRIMER 3271
PopupWindow class PRIMER 3276
Ssh class PRIMER 3278
Utils class PRIMER 3285
View class PRIMER 3293
Widget class PRIMER 3301
WidgetItem class PRIMER 3326
Window class PRIMER 3331
Workflow class PRIMER 3350
XlsxWorkbook class PRIMER 3358
XlsxWorksheet class PRIMER 3360
XMLParser class PRIMER 3363
Xrefs class PRIMER 3366
Zip class PRIMER 3369
D3PLOT **D3PLOT 1**
global class D3PLOT 1
Beam class D3PLOT 14
Colour class D3PLOT 29
Component class D3PLOT 31
Constant class D3PLOT 47
Contact class D3PLOT 51
CutSection class D3PLOT 65
File class D3PLOT 76

GraphicsWindow class D3PLOT 95
Group class D3PLOT 104
Image class D3PLOT 109
Include class D3PLOT 113
Link class D3PLOT 117
Material class D3PLOT 123
Measure class D3PLOT 130
Model class D3PLOT 137
Node class D3PLOT 144
Options class D3PLOT 159
Page class D3PLOT 160
Part class D3PLOT 165
PopupWindow class D3PLOT 181
Segment class D3PLOT 183
SetBeam class D3PLOT 197
SetNode class D3PLOT 205
SetPart class D3PLOT 213
SetShell class D3PLOT 221
SetSolid class D3PLOT 229
SetTshell class D3PLOT 237
Shell class D3PLOT 245
Solid class D3PLOT 262
Tshell class D3PLOT 279
Type class D3PLOT 296
View class D3PLOT 298
Widget class D3PLOT 302
WidgetItem class D3PLOT 327
Window class D3PLOT 332
Workflow class D3PLOT 349
XlsxWorkbook class D3PLOT 357
XlsxWorksheet class D3PLOT 359
XMLParser class D3PLOT 362
Composites D3PLOT 365
Contacts D3PLOT 368
CutSectionOld D3PLOT 370
Data D3PLOT 379
DataComponents D3PLOT 390
Elements D3PLOT 407
Groups D3PLOT 411
Includes D3PLOT 412
IntegrationPoints D3PLOT 413
Labels D3PLOT 415
Materials D3PLOT 416
Models D3PLOT 417
Parts D3PLOT 419
Selecting D3PLOT 421
Sets D3PLOT 423
SharedConstants D3PLOT 425
Ssh class D3PLOT 430
States D3PLOT 437
UserComponents D3PLOT 440
Utils class D3PLOT 447
Visibility D3PLOT 454
Windows D3PLOT 459
Examples D3PLOT 464
T/HIS **T/HIS 1**
global class T/HIS 1
Colour class T/HIS 15
Component class T/HIS 18
Constant class T/HIS 45
Curve class T/HIS 46
Datum class T/HIS 62
Entity class T/HIS 69
File class T/HIS 73
Graph class T/HIS 92
Group class T/HIS 104
Include class T/HIS 112
LineStyle class T/HIS 113
LineWidth class T/HIS 114
Link class T/HIS 115
Model class T/HIS 121

Operate class T/HIS 134
Options class T/HIS 186
Page class T/HIS 187
PopupWindow class T/HIS 194
Read class T/HIS 196
Ssh class T/HIS 203
Symbol class T/HIS 210
Units class T/HIS 211
UnitSystem class T/HIS 215
Utils class T/HIS 216
Widget class T/HIS 223
WidgetItem class T/HIS 248
Window class T/HIS 253
Workflow class T/HIS 270
XMLParser class T/HIS 278
REPORTER **REPORTER 1**
global class REPORTER 1
Colour class REPORTER 7
File class REPORTER 16
Image class REPORTER 28
Include class REPORTER 37
Item class REPORTER 38
Options class REPORTER 59
Page class REPORTER 60
Reporter class REPORTER 64
Template class REPORTER 67
Utils class REPORTER 77
Variable class REPORTER 83
Window class REPORTER 87

global class

The global class is the main JavaScript class. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- AllocateFlag()
- BatchMode()
- DialogueFunction(name*[function]*)
- DialogueInput(command*[string]*)
- DialogueInputNoEcho(command*[string]*)
- ErrorMessage(string*[Any valid javascript type]*)
- Execute(data*[object]*)
- Exit(write hook interrupt (optional)*[boolean]*)
- FlagsAvailable()
- GetCurrentDirectory()
- GetInstallDirectory()
- GetPreferenceValue(program*[string]*, name*[string]*)
- GetStartInDirectory()
- Getenv(name*[string]*)
- Labels(type*[string]*, state (optional)*[boolean]*)
- MacroFunction(name*[function]*)
- Message(string*[Any valid javascript type]*)
- MilliSleep(time*[integer]*)
- NumberToString(number*[integer/real]*, width*[integer]*, pref_int (optional)*[boolean]*)
- OpenManual(program*[string]*, page*[string]*)
- PlayMacro(filename*[string]*, options (optional)*[object]*)
- Print(string*[Any valid javascript type]*)
- Println(string*[Any valid javascript type]*)
- Requires(build*[integer]*)
- ReturnFlag(flag*[Flag]*)
- RunScript(filename*[string]*, separate (optional)*[boolean]*)
- SetCurrentDirectory(directory path*[string]*)
- SetPreferenceValue(program*[string]*, name*[string]*, value*[string]*, refresh (optional)*[boolean]*)
- Sleep(time*[integer]*)
- System(string*[Any valid javascript type]*)
- Unix()
- Use(filename*[string]*)
- UuidCreate()
- Visibility(type*[string]*, state (optional)*[boolean]*)
- WarningMessage(string*[Any valid javascript type]*)
- Windows()

Detailed Description

The global class declares the global object in JavaScript that contains the global properties and methods. As well as the core JavaScript methods, PRIMER also defines other additional ones. e.g. Message(), Print() etc. See the documentation below for more details.

Details of functions

AllocateFlag() [static]

Description

Allocate a flag for use in the script. See also ReturnFlag() and Model.PropagateFlag(). Once allocated the flag is automatically cleared for all the models currently in PRIMER.

Arguments

No arguments

Returns

Flag

Return type

Number

Example

To allocate a flag

```
var flag = AllocateFlag();
```

BatchMode() [static]

Description

Check if PRIMER is running in "batch mode" (i.e. menus are not active). Menus will not be active if PRIMER is started with the -d=tty command line argument. Note that this is different to starting PRIMER with the -batch command line argument. When using -batch, the menu system is still running, but the main PRIMER window is not shown.

Arguments

No arguments

Returns

true if in batch mode, false if not

Return type

Boolean

Example

To test if PRIMER is in batch mode

```
var batch_mode = BatchMode();
```

DialogueFunction(name*[function]*) [static]

Description

Set the function for dialogue callback. This function can be used to make PRIMER return any dialogue messages that are printed. This may be useful for you to know if a particular dialogue message has been printed or a particular event has taken place.
The function will be called with 1 argument which is a string containing the dialogue message. To remove the dialogue function use DialogueFunction(null).

Arguments

- **name** (function)

The name of the function (or null to remove the function)

Returns

No return value

Example

To set function MyDialogueFunction as the dialogue function:

```
DialogueFunction(MyDialogueFunction);
```

DialogueInput(command*[string]*) [static]

Description

Execute one or more lines of command line dialogue input.

Arguments

- **command** (string)

Command to execute (as if it had been typed into the dialogue box)

This argument can be repeated if required
Alternatively a single array argument containing the multiple values can be given

Returns

0: No errors/warnings.
> 0: This number of errors occurred.
< 0: Absolute number is the number of warnings that occurred.

Return type

Number

Example

To read two models:

```
DialogueInput("/rea dk model_1.key 1", "/rea dk model_2.key 2");
```

Note that each call to DialogueInput starts afresh at the top of the PRIMER command line "tree", so where multiple commands need to be given at sub-menu levels they need to be included in a single call. For example to restrain degrees of a mechanism assembly, and then move it by some amount:

```
DialogueInput("/mech assy " + assy_number, "fix 123", "done", "point " + point_
name, delta_x + " * *", "accept");
```

NOT:

```
DialogueInput("/mech assy " + assy_number);

DialogueInput("fix 123");
```

etc

DialogueInputNoEcho(command*[string]*) [static]

Description

Execute one or more lines of command line dialogue input **with no echo of commands to dialogue box**.

Arguments

- **command** (string)

Command to execute (as if it had been typed into the dialogue box)

This argument can be repeated if required

Alternatively a single array argument containing the multiple values can be given

Returns

0: No errors/warnings.
> 0: This number of errors occurred.
< 0: Absolute number is the number of warnings that occurred.

Return type

Number

Example

To read two models:

```
DialogueInputNoEcho("/re dk model_1.key 1", "/re dk model_2.key 2");
```

As with DialogueInput above each call starts at the top of the PRIMER command tree structure, so any commands destined for sub-menus must all be arguments to a single call.

ErrorMessage(string*[Any valid javascript type]*) [static]

Description

Print an error message to the dialogue box **adding a carriage return**.

Arguments

- **string** (Any valid javascript type)

The string/item that you want to print

Returns

No return value

Example

To print the title of model object m as an error to the dialogue box

```
ErrorMessage("The title is " + m.title);
```

Execute(data*[object]*) [static]

Description

Execute a program or script outside PRIMER and get the standard output and error streams.

Arguments

- **data** (object)

Execute data

Object has the following properties:

Name	Type	Description
arguments (optional)	Array of strings	The arguments to pass to program
program	string	The program you want to run. Note that on Linux this will consider PATH when resolving executable filenames without an absolute path. If you want to run something from the current directory and you do not have '.' in your PATH then you will need to write './something' as the program.

Returns

Object with the following properties:

Name	Type	Description
status	integer	The exit code from the program/script
stderr	string	The standard error output from the program/script
stdout	string	The standard output from the program/script

Return type

object

Example

To run script "example.bat" with arguments "foo" and "bar":

```
var output = Execute( { program: 'example.bat', arguments: [ 'foo', 'bar' ]} );
var text   = output.stdout;
var errors = output.stderr;
var ecode  = output.status;
```

Exit(write hook interrupt (optional)*[boolean]*) [static]

Description

Exit script

Arguments

- **write hook interrupt (optional)** (boolean)

If Exit() is called from a write_hook.js script, the first argument will be processed as in the following: If the argument is provided and set to "true", it is used to interrupt the write out of the model, so that the script exits without anything being written out. An argument value of "false" exits the script and allows the model to be written out as normal. An example of this function's use in a Write Hook script can be found at $OA_INSTALL/primer_library/scripts/hooks/example_write_hook.js.

Returns

No return value

Example

Exit with

```
Exit();
```

FlagsAvailable() [static]

Description

Number of flags available to be used for AllocateFlag()

Arguments

No arguments

Returns

Number of flags available

Return type

Number

Example

To get the number of flags available:

```
var flags = FlagsAvailable();
```

GetCurrentDirectory() [static]

Description

Get the current working directory

Arguments

No arguments

Returns

String containing current working directory

Return type

String

Example

To get the current directory:

```
var cwd = GetCurrentDirectory();
```

GetInstallDirectory() [static]

Description

Get the directory in which executables are installed. This is the OA_INSTALL environment variable, or if that is not set the directory in which the current executable is installed. Returns NULL if not found

Arguments

No arguments

Returns

string

Return type

String

Example

To get the install directory:

```
var install_dir = GetInstallDirectory();
```

GetPreferenceValue(program*[string]*, name*[string]*) [static]

Description

Get the Preference value with the given string in the any of admin ("OA_ADMIN") or install ("OA_INSTALL") or home ("OA_HOME") directory oa_pref

Arguments

- **program** (string)

The program name string : Valid values are ’All’, ’D3PLOT’, ’PRIMER’, ’REPORTER’, ’SHELL’, ’T/HIS’

- **name** (string)

The preference name string

Returns

: String containing preference value or null if preference string is not present in any oa_pref. Also if none of the above environment variables are not present, then API simply returns null. While returning preference value, locked preference value in admin and then install oa_pref takes precedence over home oa_pref. If preference is not locked in any of these oa_pref, preference in home directory oa_pref is returned.

Return type

String

Example

To get the preference value:

```
var pref_list = GetPreferenceValue(’All’, "font_size");
```

GetStartInDirectory() [static]

Description

Get the directory passed to PRIMER by the -start_in command line argument

Arguments

No arguments

Returns

String containing start_in directory or NULL if not set

Return type

String

Example

To get the start_in directory:

```
var start_in = GetStartInDirectory();
```

Getenv(name*[string]*) [static]

Description

Get the value of an environment variable

Arguments

- **name** (string)

The environment variable name

Returns

String containing variable value or null if variable does not exist

Return type

String

Example

To get the value for environment variable HOME

```
var home = Getenv("HOME");
```

Labels(type*[string]*, state (optional)*[boolean]*) [static]

Description

Set or get labelling of items in PRIMER

Arguments

- **type** (string)

The type of the item (for a list of types see Appendix I of the PRIMER manual). Additionally, to change the visibility of attached or unattached nodes you can use the types "ATTACHED_NODE" and "UNATTACHED_NODE".

- **state (optional)** (boolean)

If it is provided it is used to set the labelling status of entity. "true" to make items labelled and "false" to make them not labelled.

Returns

Boolean

Return type

Boolean

Example

To turn on beam labels

```
Labels("BEAM", true);
```

To get the labelling status of beams

```
var lab = Labels("BEAM");
```

MacroFunction(name*[function]*) [static]

Description

Set the function for macro callback. This function can be used to make PRIMER return the macro command that would be recorded if macro recording was active for every button press etc. This may be useful for you to know if a particular action has been done by the user.
The function will be called with 1 argument which is a string containing the macro command. To remove the macro function use MacroFunction(null).

Arguments

- **name** (function)

The name of the function (or null to remove a function)

Returns

No return value

Example

To set function MyMacroFunction as the macro function:

```
MacroFunction(MyMacroFunction);
```

Message(string*[Any valid javascript type]*) [static]

Description

Print a message to the dialogue box **adding a carriage return**.

Arguments

- **string** (Any valid javascript type)

The string/item that you want to print. If '\r' is added to the end of the string then instead of automatically adding a carriage return in the dialogue box, the next message will overwrite the current one. This may be useful for giving feedback to the dialogue box when doing an operation.

Returns

No return value

Example

To print the title of model object m as a message to the dialogue box

```
Message("The title is " + m.title);
```

MilliSleep(time*[integer]*) [static]

Description

Pause execution of the script for *time* milliseconds. See also Sleep()

Arguments

- **time** (integer)

Number of milliseconds to pause for

Returns

No return value

Example

To pause for 500 milliseconds

```
MilliSleep(500);
```

NumberToString(number*[integer/real]*, width*[integer]*, pref_int (optional)*[boolean]*) [static]

Description

Formats a number to a string with the specified width.

Arguments

- **number** (integer/real)

The number you want to format.

- **width** (integer)

The width of the string you want to format it to (must be less than 80).

- **pref_int (optional)** (boolean)

By default only integer values inside the single precision 32 bit signed integer limit of approximately +/-2e9 are formatted as integers, all other numeric values are formatted as floats. With this argument set to TRUE then integer values up to the mantissa precision of a 64 bit float, approximately +/-9e15, will also be formatted as integers.

Returns

String containing the number

Return type

String

Example

To write the number 1.2345e+6 to a string 10 characters wide

```
var str = NumberToString(1.2345e+6, 10);
```

OpenManual(program*[string]*, page*[string]*) [static]

Description

Open the Oasys manuals at a requested page

Arguments

- **program** (string)

The program manual to open. Can be "primer", "d3plot" or "this"

- **page** (string)

The page to open in the manual, e.g. "running-this.html"

Returns

true if successful, false if not

Return type

Boolean

Example

To open the T/HIS manual on the running-this.html page

```
OpenManual("this", "running-this.html");
```

PlayMacro(filename*[string]*, options (optional)*[object]*) [static]

Description

Play a macro in PRIMER

Arguments

- **filename** (string)

The name of the macro file to play

- **options (optional)** (object)

Options specifying how the macro file should be replayed. If omitted the default values below will be used.

Object has the following properties:

Name	Type	Description
delay	integer	Delay in ms between commands when replaying. If omitted the current value from macro window will be used.
pick	boolean	If picks/drags from the macro file should be replayed. If omitted the current value from macro window will be used.
terminate	boolean	If the script should be terminated if an error occurs when playing the macro. If omitted the script will be terminated.
utf8	boolean	If the script is UTF-8 encoded. If omitted or false the script is assumed to be ASCII text.
variables	object	Object containing names and values for variables in the macro. If null or omitted no variables are used.
view	boolean	If views encoded in the macro file for picks/drags should be replayed. If omitted the current value from macro window will be used.

Returns

true if an error occured during playback, false otherwise.

Return type

Boolean

Example

To play a UTF-8 encoded macro file /data/test/example.prm using the default options for picking/dragging and a delay of 500ms:

```
PlayMacro("/data/test/example.prm", { delay:500, utf8:true} );
```

To play macro file /data/test/example.prm, defining values for variables A, B and C in the macro:

```
PlayMacro("/data/test/example.prm", { variables: { A:10.0, B:0, C:"Example"} }
);
```

Print(string*[Any valid javascript type]*) [static]

Description

Print a string to stdout. **Note that a carriage return is not added**.

Arguments

- **string** (Any valid javascript type)

The string/item that you want to print

Returns

No return value

Example

To print string "Hello, world!"

```
Print("Hello, world!");
```

To print the title of model object m with a carriage return

```
print("The title is " + m.title + "\n");
```

Println(string*[Any valid javascript type]*) [static]

Description

Print a string to stdout **adding a carriage return**.

Arguments

- **string** (Any valid javascript type)

The string/item that you want to print

Returns

No return value

Example

To print string "Hello, world!" automatically adding a carriage return

```
Println("Hello, world!");
```

To print the title of model object m, automatically adding a carriage return

```
Println("The title is " + m.title);
```

Requires(build*[integer]*) [static]

Description

Checks to see if the build number of PRIMER is high enough to run this script. If your script requires features that are only present in builds of PRIMER greater than a certain value Require can test this and only run the script if the build is high enough.

Arguments

- **build** (integer)

The minimum build number that is required.

Returns

No return value (if the build is not high enough the script will terminate)

Example

To only allow a script to run if the build is >= 2000

```
Requires(2000);
```

ReturnFlag(flag*[Flag]*) [static]

Description

Return a flag used in the script. See also AllocateFlag() and Model.PropagateFlag().

Arguments

- **flag** (Flag)

The flag to return.

Returns

No return value.

Example

To return flag f:

```
ReturnFlag(f);
```

RunScript(filename*[string]*, separate (optional)*[boolean]*) [static]

Description

Run a script. Note: RunScript is intended to run a 'child' script that will finish before the calling script finishes. Terminating the calling script while child scripts are still running may give undefined behaviour.

Arguments

- **filename** (string)

The name of the script file to run. If the filename is relative then the file will be searched for relative to this script. If not found then the script_directory preference will be used.

- **separate (optional)** (boolean)

If the script will use separate memory from the current script. If it uses separate memory (true) then the 'child' script is completely separated from this script and knows nothing about variables in this script. If it does not use separate memory (false) then the 'child' script will have access to all of the variables in the current script and hence variables must not clash. It is strongly recommended that you use namespaces to stop variable names from clashing. If omitted the script will use separate memory.

Returns

No return value

Example

To run script /data/test/child.js using separate memory for the child script

```
RunScript("/data/test/child.js");
```

SetCurrentDirectory(directory path*[string]*) [static]

Description

Sets the current working directory.

Arguments

- **directory path** (string)

Path to the directory you would like to change into.

Returns

true if successful, false if not

Return type

Boolean

Example

To change into the directory "/data/test" exists

```
SetCurrentDirectory("/data/test")
```

SetPreferenceValue(program*[string]*, name*[string]*, value*[string]*, refresh (optional)*[boolean]*) [static]

Description

Save the preference string and its value into oa_pref of home directory. If the preference is locked in admin ("OA_ADMIN") or install ("OA_INSTALL") oa_pref, then API is unsuccessful. Home directory is defined by environment variable OA_HOME. If OA_HOME is not defined then API is unsuccessful.

Arguments

- **program** (string)

The program name string : Valid values are ’All’, ’D3PLOT’, ’PRIMER’, ’REPORTER’, ’SHELL’, ’T/HIS’

- **name** (string)

The preference name string

- **value** (string)

The preference value string. If "value" is of zero length, then the option is simply removed from the file if present, and no new entry is made.This argument cannot be null.

- **refresh (optional)** (boolean)

If the saved preference should be refreshed. If omitted, the preference will NOT be refreshed. This argument is currently only available in PRIMER JS API and ignored in D3PLOT and T/HIS.

Returns

An integer. Returns 0 if the preference is saved succesfully or 1 if unsuccessful

Return type

Number

Example

To save the preference value:

```
var ierr = SetPreferenceValue( ’All’, "font_size", ’Default’);
```

Sleep(time*[integer]*) [static]

Description

Pause execution of the script for *time* seconds. See also MilliSleep()

Arguments

- **time** (integer)

Number of seconds to pause for

Returns

No return value

Example

To pause for 2 seconds

```
Sleep(2);
```

System(string*[Any valid javascript type]*) [static]

Description

Do a system command outside PRIMER. To run an external command and get the output then please use Execute() instead.

Arguments

- **string** (Any valid javascript type)

The system command that you want to do

Returns

integer (probably zero if command successful but is implementation-dependant)

Return type

Number

Example

To make the directory "example"

```
System("mkdir example");
```

Unix() [static]

Description

Test whether script is running on a Unix/Linux operating system. See also Windows()

Arguments

No arguments

Returns

true if Unix/Linux, false if not

Return type

Boolean

Example

To test if the OS is Unix

```
if ( Unix() )
```

Use(filename*[string]*) [static]

Description

Use script from a separate file

Arguments

- **filename** (string)

Use allows you to include a script from a separate file. This may be useful if your script is very large and you want to split it up to help with maintenance. Alternatively you may have a 'library' of common functions which you always want to include in your scripts. Including the 'library' with Use means that any changes only have to be done in one place. PRIMER will look for the file in the same directory as the main script. If that fails then it will look in $OA_INSTALL/primer_library/scripts directory and the script directory specified by the *primer*script_directory* preference. **Note that the file is included when the script is compiled, NOT at runtime**.

Returns

No return value

Example

To include script from file library.js

```
Use("library.js");
```

UuidCreate() [static]

Description

Create a UUID (Universally unique ID)

Arguments

No arguments

Returns

string

Return type

String

Example

To create a UUID:

```
var uuid = UuidCreate();
```

Visibility(type*[string]*, state (optional)*[boolean]*) [static]

Description

Set or get visibility of items in PRIMER

Arguments

- **type** (string)

The type of the item (for a list of types see Appendix I of the PRIMER manual). Additionally, to change the visibility of attached or unattached nodes you can use the types "ATTACHED_NODE" and "UNATTACHED_NODE".

- **state (optional)** (boolean)

If it is provided it is used to set the visibility. "true" to make items visible and "false" to make them not visible.

Returns

Boolean

Return type

Boolean

Example

To make beams visible

```
Visibility("BEAM", true);
```

To get the visibility status of beams

```
var vis = Visibility("BEAM");
```

WarningMessage(string*[Any valid javascript type]*) [static]

Description

Print a warning message to the dialogue box **adding a carriage return**.

Arguments

- **string** (Any valid javascript type)

The string/item that you want to print

Returns

No return value

Example

To print the title of model object m as a warning to the dialogue box

```
WarningMessage("The title is " + m.title);
```

Windows() [static]

Description

Test whether script is running on a Windows operating system. See also Unix()

Arguments

No arguments

Returns

true if Windows, false if not

Return type

Boolean

Example

To test if the OS is Windows

```
if ( Windows() )
```

Airbag class

The Airbag class gives you access to airbag cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- ExtractColour()
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- GetPropertyByIndex(index*[integer]*)
- GetPropertyByName(acronym*[string]*)
- GetPropertyByRowCol(row*[integer]*, col*[integer]*)
- GetPropertyNameForIndex(index*[integer]*)
- GetPropertyNameForRowCol(row*[integer]*, col*[integer]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- SetPropertyByIndex(index*[integer]*, value*[integer/real for numeric properties, string for character properties]*)
- SetPropertyByName(acronym*[string]*, value*[integer/real for numeric properties, string for character properties]*)
- SetPropertyByRowCol(row*[integer]*, col*[integer]*, value*[integer/real for numeric properties, string for character properties]*)
- Sketch(redraw (optional)*[boolean]*)
- Unsketch(redraw (optional)*[boolean]*)

- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

Airbag constants

Name	Description
Airbag.ADIABATIC_GAS_MODEL	Airbag adiabatic gas model type
Airbag.ADVANCED_ALE	Airbag advanced ALE type
Airbag.ALE	Airbag ALE type
Airbag.CPG	Airbag CPG type
Airbag.HYBRID	Airbag hybrid type
Airbag.HYBRID_CHEMKIN	Airbag hybrid chemkin type
Airbag.HYBRID_JETTING	Airbag hybrid jetting type
Airbag.LINEAR_FLUID	Airbag linear fluid type
Airbag.LOAD_CURVE	Airbag load curve type
Airbag.PARTICLE	Airbag particle type
Airbag.SALE	Airbag SALE type
Airbag.SIMPLE_AIRBAG_MODEL	Airbag simple airbag model type
Airbag.SIMPLE_PRESSURE_VOLUME	Airbag simple pressure volume type
Airbag.WANG_NEFSKE	Airbag Wang Nefske type
Airbag.WANG_NEFSKE_JETTING	Airbag Wang Nefske jetting type
Airbag.WANG_NEFSKE_MULTIPLE_JETTING	Airbag Wang Nefske multiple jetting type

Airbag properties

Name	Type	Description
abid	integer	Airbag number. Also see the label property which is an alternative name for this.
colour	Colour	The colour of the airbag
cols (read only)	real	The number of columns of data the airbag has
exists (read only)	logical	true if airbag exists, false if referred to but not defined.
id	logical	Turns _ID on or OFF
include	integer	The Include file number that the airbag is in.
label	integer	Airbag number. Also see the abid property which is an alternative name for this.
model (read only)	integer	The Model number that the airbag is in.
properties	integer	The total number of properties that the airbag has
rows (read only)	integer	The number of rows of data the airbag has. This includes the _ID card if it is set.

title	string	Airbag title
type	constant	Airbag type. Can be Airbag.SIMPLE_PRESSURE_VOLUME, Airbag.SIMPLE_AIRBAG_MODEL, Airbag.ADIABATIC_GAS_MODEL, Airbag.WANG_NEFSKE, Airbag.WANG_NEFSKE_JETTING, Airbag.WANG_NEFSKE_MULTIPLE_JETTING, Airbag.LOAD_CURVE, Airbag.LINEAR_FLUID, Airbag.HYBRID, Airbag.HYBRID_JETTING, Airbag.HYBRID_CHEMKIN, Airbag.ALE, Airbag.ADVANCED_ALE, Airbag.PARTICLE, Airbag.CPG or Airbag.SALE

Detailed Description

The Airbag class allows you to create, modify, edit and manipulate airbag cards. See the documentation below for more details.

Constructor

new Airbag(Model*[Model]*, type*[string]*, sid*[integer]*, sidtyp (optional)*[integer]*, abid (optional)*[integer]*, heading (optional)*[string]*)

Description

Create a new Airbag object.

Arguments

- **Model** (Model)

Model that airbag will be created in

- **type** (string)

Airbag type. Can be Airbag.SIMPLE_PRESSURE_VOLUME, Airbag.SIMPLE_AIRBAG_MODEL, Airbag.ADIABATIC_GAS_MODEL, Airbag.WANG_NEFSKE, Airbag.WANG_NEFSKE_JETTING, Airbag.WANG_NEFSKE_MULTIPLE_JETTING, Airbag.LOAD_CURVE, Airbag.LINEAR_FLUID, Airbag.HYBRID, Airbag.HYBRID_JETTING, Airbag.HYBRID_CHEMKIN, Airbag.ALE, Airbag.ADVANCED_ALE, Airbag.PARTICLE, Airbag.CPG or Airbag.SALE

- **sid** (integer)

Set ID

- **sidtyp (optional)** (integer)

Set type: segment/part set ID

- **abid (optional)** (integer)

Airbag number

- **heading (optional)** (string)

Airbag title

Returns

Airbag object

Return type

Airbag

Example

To create a new AIRBAG_SIMPLE_PRESSURE_VOLUME in model m with set ID 10 and segment set type

```
var a = new Airbag(m, Airbag.SIMPLE_PRESSURE_VOLUME, 10);
```

or

```
var a = new Airbag(m, Airbag.SIMPLE_PRESSURE_VOLUME, 10, 0);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a airbag.

Arguments

- **Comment** (Comment)

Comment that will be attached to the airbag

Returns

No return value

Example

To associate comment c to the airbag a:

```
a.AssociateComment(c);
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse airbag a:

```
a.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the airbag.

Arguments

- **flag** (Flag)

Flag to clear on the airbag

Returns

No return value

Example

To clear flag f for airbag a:

```
a.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the airbag. The target include of the copied airbag can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

Airbag object

Return type

Airbag

Example

To copy airbag a into airbag z:

```
var z = a.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a airbag

Arguments

- **Model** (Model)

Model that the airbag will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

Airbag object (or null if not made).

Return type

Airbag

Example

To start creating a airbag in model m:

```
var a = Airbag.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a airbag.

Arguments

- **Comment** (Comment)

Comment that will be detached from the airbag

Returns

No return value

Example

To detach comment c from the airbag a:

```
a.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit airbag a:

```
a.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for airbag with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description

autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The airbag object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for airbag a:

```
a.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for airbag a:

```
a.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for airbag. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for airbag a:

```
a.Error("My custom error");
```

ExtractColour()

Description

Extracts the **actual** colour used for airbag.
By default in PRIMER many entities such as elements get their colour automatically from the part that they are in. PRIMER cycles through 13 default colours based on the label of the entity. In this case the airbag colour property will return the value Colour.PART instead of the actual colour. This method will return the actual colour which is used for drawing the airbag.

Arguments

No arguments

Returns

colour value (integer)

Return type

Number

Example

To return the colour used for drawing airbag a:

```
var colour = a.ExtractColour();
```

First(Model*[Model]*) [static]

Description

Returns the first airbag in the model.

Arguments

- **Model** (Model)

Model to get first airbag in

Returns

Airbag object (or null if there are no airbags in the model).

Return type

Airbag

Example

To get the first airbag in model m:

```
var a = Airbag.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free airbag label in the model. Also see Airbag.LastFreeLabel(), Airbag.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free airbag label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

Airbag label.

Return type

Number

Example

To get the first free airbag label in model m:

```
var label = Airbag.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the airbags in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all airbags will be flagged in

- **flag** (Flag)

Flag to set on the airbags

Returns

No return value

Example

To flag all of the airbags with flag f in model m:

```
Airbag.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the airbag is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the airbag

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if airbag a has flag f set on it:

```
if (a.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each airbag in the model.
Note that ForEach has been designed to make looping over airbags as fast as possible and so has some limitations.
Firstly, a single temporary Airbag object is created and on each function call it is updated with the current airbag data. This means that you should not try to store the Airbag object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new airbags inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all airbags are in

- **func** (function)

Function to call for each airbag

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the airbags in model m:

```
Airbag.ForEach(m, test);
function test(a)
{
// a is Airbag object
}
```

To call function test for all of the airbags in model m with optional object:

```
var data = { x:0, y:0 };
Airbag.ForEach(m, test, data);
function test(a, extra)
{
// a is Airbag object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Airbag objects or properties for all of the airbags in a model in PRIMER. If the optional property argument is not given then an array of Airbag objects is returned. If the property argument is given, that property value for each airbag is returned in the array instead of a Airbag object

Arguments

- **Model** (Model)

Model to get airbags from

- **property (optional)** (string)

Name for property to get for all airbags in the model

Returns

Array of Airbag objects or properties

Return type

Array

Example

To make an array of Airbag objects for all of the airbags in model m:

```
var a = Airbag.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each airbag in model m:

```
var a = Airbag.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a airbag.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the airbag a:

```
var comm_array = a.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of Airbag objects for all of the flagged airbags in a model in PRIMER If the optional property argument is not given then an array of Airbag objects is returned. If the property argument is given, then that property value for each airbag is returned in the array instead of a Airbag object

Arguments

- **Model** (Model)

Model to get airbags from

- **flag** (Flag)

Flag set on the airbags that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged airbags in the model

Returns

Array of Airbag objects or properties

Return type

Array

Example

To make an array of Airbag objects for all of the airbags in model m flagged with f:

```
var a = Airbag.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the airbags in model m flagged with f:

```
var a = Airbag.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the Airbag object for a airbag ID.

Arguments

- **Model** (Model)

Model to find the airbag in

- **number** (integer)

number of the airbag you want the Airbag object for

Returns

Airbag object (or null if airbag does not exist).

Return type

Airbag

Example

To get the Airbag object for airbag 100 in model m

```
var a = Airbag.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a Airbag property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the Airbag.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

airbag property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if Airbag property a.example is a parameter:

```
Options.property_parameter_names = true;
if (a.GetParameter(a.example) ) do_something...
Options.property_parameter_names = false;
```

To check if Airbag property a.example is a parameter by using the GetParameter method:

```
if (a.ViewParameters().GetParameter(a.example) ) do_something...
```

GetPropertyByIndex(index*[integer]*)

Description

Returns the value of property at index *index* for this Airbag object or null if no property exists.

Arguments

- **index** (integer)

The index of the property value to retrieve. (the number of properties can be found from properties) **Note that indices start at 0**. There is no link between indices and rows/columns so adjacent fields on a line for an airbag may not have adjacent indices.

Returns

Property value (real/integer)

Return type

Number

Example

To return the property at index 3, for airbag a:

```
var prop = a.GetPropertyByIndex(3);
```

GetPropertyByName(acronym*[string]*)

Description

Returns the value of property string *acronym* for this Airbag object or null if no property exists.

Arguments

- **acronym** (string)

The acronym of the property value to retrieve

Returns

Property value (real/integer)

Return type

Number

Example

To return the value of HCONV for airbag a:

```
var hconv = a.GetPropertyByName("HCONV");
```

GetPropertyByRowCol(row*[integer]*, col*[integer]*)

Description

Returns the value of the property for row and col for this Airbag object or null if no property exists. **Note that columns start at 0. Rows start at 1 if the _ID option is set, at 0 otherwise**.

Arguments

- **row** (integer)

The row of the property value to retrieve

- **col** (integer)

The column of the property value to retrieve

Returns

Property value (real/integer)

Return type

Number

Example

To return the value of the property at row 0, column 3 for airbag a:

```
var prop = a.GetPropertyByRowCol(0, 3);
```

GetPropertyNameForIndex(index*[integer]*)

Description

Returns the name of the property at index *index* for this Airbag object or null if there is no property.

Arguments

- **index** (integer)

The index of the property name to retrieve. (the number of properties can be found from properties) **Note that indices start at 0**. There is no link between indices and rows/columns so adjacent fields on a line for an airbag may not have adjacent indices.

Returns

Property name (string)

Return type

String

Example

To return the name of the property at index 3, for airbag a:

```
var name = a.GetPropertyNameForIndex(3);
```

GetPropertyNameForRowCol(row*[integer]*, col*[integer]*)

Description

Returns the name of the property at row and col for this Airbag object or null if there is no property. **Note that columns start at 0. Rows start at 1 if the _ID option is set, at 0 otherwise**.

Arguments

- **row** (integer)

The row of the property name to retrieve

- **col** (integer)

The column of the property name to retrieve

Returns

Property name (string)

Return type

String

Example

To return the name of the property at row 0, column 1 for airbag a:

```
var name = a.GetPropertyNameForRowCol(0, 1);
```

Keyword()

Description

Returns the keyword for this airbag (e.g. *AIRBAG_SIMPLE_PRESSURE_VOLUME, *AIRBAG_SIMPLE_AIRBAG_MODEL etc). **Note that a carriage return is not added**. See also Airbag.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for airbag a:

```
var key = a.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the airbag. **Note that a carriage return is not added**. See also Airbag.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for airbag a:

```
var cards = a.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last airbag in the model.

Arguments

- **Model** (Model)

Model to get last airbag in

Returns

Airbag object (or null if there are no airbags in the model).

Return type

Airbag

Example

To get the last airbag in model m:

```
var a = Airbag.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free airbag label in the model. Also see Airbag.FirstFreeLabel(), Airbag.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free airbag label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

Airbag label.

Return type

Number

Example

To get the last free airbag label in model m:

```
var label = Airbag.LastFreeLabel(m);
```

Next()

Description

Returns the next airbag in the model.

Arguments

No arguments

Returns

Airbag object (or null if there are no more airbags in the model).

Return type

Airbag

Example

To get the airbag in model m after airbag a:

```
var a = a.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) airbag label in the model. Also see Airbag.FirstFreeLabel(), Airbag.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free airbag label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

Airbag label.

Return type

Number

Example

To get the next free airbag label in model m:

```
var label = Airbag.NextFreeLabel(m);
```

Previous()

Description

Returns the previous airbag in the model.

Arguments

No arguments

Returns

Airbag object (or null if there are no more airbags in the model).

Return type

Airbag

Example

To get the airbag in model m before airbag a:

```
var a = a.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the airbags in the model.

Arguments

- **Model** (Model)

Model that all airbags will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the airbags in model m, from 1000000:

```
Airbag.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged airbags in the model.

Arguments

- **Model** (Model)

Model that all the flagged airbags will be renumbered in

- **flag** (Flag)

Flag set on the airbags that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the airbags in model m flagged with f, from 1000000:

```
Airbag.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select airbags using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting airbags

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only airbags from that model can be selected. If the argument is a Flag then only airbags that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any airbags can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of airbags selected or null if menu cancelled

Return type

Number

Example

To select airbags from model m, flagging those selected with flag f, giving the prompt 'Select airbags':

```
Airbag.Select(f, 'Select airbags', m);
```

To select airbags, flagging those selected with flag f but limiting selection to airbags flagged with flag l, giving the prompt 'Select airbags':

```
Airbag.Select(f, 'Select airbags', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the airbag.

Arguments

- **flag** (Flag)

Flag to set on the airbag

Returns

No return value

Example

To set flag f for airbag a:

```
a.SetFlag(f);
```

SetPropertyByIndex(index*[integer]*, value*[integer/real for numeric properties, string for character properties]*)

Description

Sets the value of property at index *index* for this Airbag object

Arguments

- **index** (integer)

The index of the property value to set. (the number of properties can be found from properties) **Note that indices start at 0**. There is no link between indices and rows/columns so adjacent fields on a line for an airbag may not have adjacent indices.

- **value** (integer/real for numeric properties, string for character properties)

The value of the property to set.

Returns

No return value

Example

To set the property at index 3, for airbag a to be 1.234:

```
a.SetPropertyByIndex(3, 1.234);
```

SetPropertyByName(acronym*[string]*, value*[integer/real for numeric properties, string for character properties]*)

Description

Sets the value of property string *acronym* for this Airbag object

Arguments

- **acronym** (string)

The acronym of the property value to set

- **value** (integer/real for numeric properties, string for character properties)

The value of the property to set.

Returns

No return value

Example

To set the value of HCONV for airbag a to be 1.23:

```
a.SetPropertyByName("HCONV", 1.23);
```

SetPropertyByRowCol(row*[integer]*, col*[integer]*, value*[integer/real for numeric properties, string for character properties]*)

Description

Sets the value of the property for row and col for this Airbag object. **Note that columns start at 0. Rows start at 1 if the _ID option is set, at 0 otherwise**.

Arguments

- **row** (integer)

The row of the property value to set

- **col** (integer)

The column of the property value to set

- **value** (integer/real for numeric properties, string for character properties)

The value of the property to set.

Returns

No return value

Example

To set the value of the property at row 0, column 3 for airbag a to be 0.5:

```
a.SetPropertyByRowCol(0, 3, 0.5);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the airbag. The airbag will be sketched until you either call Airbag.Unsketch(), Airbag.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the airbag is sketched. If omitted redraw is true. If you want to sketch several airbags and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch airbag a:

```
a.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged airbags in the model. The airbags will be sketched until you either call Airbag.Unsketch(), Airbag.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged airbags will be sketched in

- **flag** (Flag)

Flag set on the airbags that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the airbags are sketched. If omitted redraw is true. If you want to sketch flagged airbags several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all airbags flagged with flag in model m:

```
Airbag.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of airbags in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing airbags should be counted. If false or omitted referenced but undefined airbags will also be included in the total.

Returns

number of airbags

Return type

Number

Example

To get the total number of airbags in model m:

```
var total = Airbag.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the airbags in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all airbags will be unset in

- **flag** (Flag)

Flag to unset on the airbags

Returns

No return value

Example

To unset the flag f on all the airbags in model m:

```
Airbag.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the airbag.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the airbag is unsketched. If omitted redraw is true. If you want to unsketch several airbags and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch airbag a:

```
a.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all airbags.

Arguments

- **Model** (Model)

Model that all airbags will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the airbags are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all airbags in model m:

```
Airbag.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged airbags in the model.

Arguments

- **Model** (Model)

Model that all airbags will be unsketched in

- **flag** (Flag)

Flag set on the airbags that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the airbags are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all airbags flagged with flag in model m:

```
Airbag.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

Airbag object.

Return type

Airbag

Example

To check if Airbag property a.example is a parameter by using the Airbag.GetParameter() method:

```
if (a.ViewParameters().GetParameter(a.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for airbag with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description

autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The airbag object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for airbag a:

```
a.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for airbag a:

```
a.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for airbag. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for airbag a:

```
a.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this airbag.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for airbag a:

```
var xrefs = a.Xrefs();
```

---

toString()

Description

Creates a string containing the airbag data in keyword format. Note that this contains the keyword header and the keyword cards. See also Airbag.Keyword() and Airbag.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for airbag a in keyword format

```
var s = a.toString();
```

ReferenceGeometry class

The ReferenceGeometry class gives you access to define airbag reference geometry cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetNode(nid*[integer]*)
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- RemoveNode(nid*[integer]*)
- SetFlag(flag*[Flag]*)
- SetNode(nid*[integer]*, x*[real]*, y*[real]*, z*[real]*)
- Sketch(redraw (optional)*[boolean]*)
- Spool()
- StartSpool()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

ReferenceGeometry properties

Name	Type	Description
aid	integer	ReferenceGeometry number. Also see the label property which is an alternative name for this.
birth	logical	Turns _BIRTH on or off
birth_time	real	Birth time
exists (read only)	logical	true if airbag reference geometry exists, false if referred to but not defined.
id	logical	Turns _ID on or OFF
include	integer	The Include file number that the airbag reference geometry is in.
iout	integer	Flag for outputting the current reference node coordinates
label	integer	ReferenceGeometry number. Also see the aid property which is an alternative name for this.
model (read only)	integer	The Model number that the airbag reference geometry is in.
nido	integer	Node number for origin
rdt	logical	Turns _RDT on or OFF
sx	real	Scale factor in X direction
sy	real	Scale factor in Y direction
sz	real	Scale factor in Z direction

Detailed Description

The ReferenceGeometry class allows you to create, modify, edit and manipulate airbag reference geometry cards. See the documentation below for more details.

Constructor

new ReferenceGeometry(Model*[Model]*, aid (optional)*[integer]*)

Description

Create a new ReferenceGeometry object.

Arguments

- **Model** (Model)

Model that ReferenceGeometry will be created in

- **aid (optional)** (integer)

ReferenceGeometry number to set _ID suffix

Returns

ReferenceGeometry object

Return type

ReferenceGeometry

Example

To create a new ReferenceGeometry in model m

```
var a = new ReferenceGeometry(m);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a airbag reference geometry.

Arguments

- **Comment** (Comment)

Comment that will be attached to the airbag reference geometry

Returns

No return value

Example

To associate comment c to the airbag reference geometry a:

```
a.AssociateComment(c);
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse airbag reference geometry a:

```
a.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the airbag reference geometry.

Arguments

- **flag** (Flag)

Flag to clear on the airbag reference geometry

Returns

No return value

Example

To clear flag f for airbag reference geometry a:

```
a.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the airbag reference geometry. The target include of the copied airbag reference geometry can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

ReferenceGeometry object

Return type

ReferenceGeometry

Example

To copy airbag reference geometry a into airbag reference geometry z:

```
var z = a.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a airbag reference geometry

Arguments

- **Model** (Model)

Model that the airbag reference geometry will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

ReferenceGeometry object (or null if not made).

Return type

ReferenceGeometry

Example

To start creating a airbag reference geometry in model m:

```
var a = ReferenceGeometry.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a airbag reference geometry.

Arguments

- **Comment** (Comment)

Comment that will be detached from the airbag reference geometry

Returns

No return value

Example

To detach comment c from the airbag reference geometry a:

```
a.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit airbag reference geometry a:

```
a.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for airbag reference geometry with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description

autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The airbag reference geometry object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for airbag reference geometry a:

```
a.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for airbag reference geometry a:

```
a.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for airbag reference geometry. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for airbag reference geometry a:

```
a.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first airbag reference geometry in the model.

Arguments

- **Model** (Model)

Model to get first airbag reference geometry in

Returns

ReferenceGeometry object (or null if there are no airbag reference geometrys in the model).

Return type

ReferenceGeometry

Example

To get the first airbag reference geometry in model m:

```
var a = ReferenceGeometry.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free airbag reference geometry label in the model. Also see ReferenceGeometry.LastFreeLabel(), ReferenceGeometry.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free airbag reference geometry label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

ReferenceGeometry label.

Return type

Number

Example

To get the first free airbag reference geometry label in model m:

```
var label = ReferenceGeometry.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the airbag reference geometrys in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all airbag reference geometrys will be flagged in

- **flag** (Flag)

Flag to set on the airbag reference geometrys

Returns

No return value

Example

To flag all of the airbag reference geometrys with flag f in model m:

```
ReferenceGeometry.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the airbag reference geometry is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the airbag reference geometry

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if airbag reference geometry a has flag f set on it:

```
if (a.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each airbag reference geometry in the model.
Note that ForEach has been designed to make looping over airbag reference geometrys as fast as possible and so has some limitations.
Firstly, a single temporary ReferenceGeometry object is created and on each function call it is updated with the current airbag reference geometry data. This means that you should not try to store the ReferenceGeometry object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new airbag reference geometrys inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all airbag reference geometrys are in

- **func** (function)

Function to call for each airbag reference geometry

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the airbag reference geometrys in model m:

```
ReferenceGeometry.ForEach(m, test);
function test(a)
{
// a is ReferenceGeometry object
}
```

To call function test for all of the airbag reference geometrys in model m with optional object:

```
var data = { x:0, y:0 };
ReferenceGeometry.ForEach(m, test, data);
function test(a, extra)
{
// a is ReferenceGeometry object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of ReferenceGeometry objects or properties for all of the airbag reference geometrys in a model in PRIMER. If the optional property argument is not given then an array of ReferenceGeometry objects is returned. If the property argument is given, that property value for each airbag reference geometry is returned in the array instead of a ReferenceGeometry object

Arguments

- **Model** (Model)

Model to get airbag reference geometrys from

- **property (optional)** (string)

Name for property to get for all airbag reference geometrys in the model

Returns

Array of ReferenceGeometry objects or properties

Return type

Array

Example

To make an array of ReferenceGeometry objects for all of the airbag reference geometrys in model m:

```
var a = ReferenceGeometry.GetAll(m);
```

To return an array containing the value of property ’foo’ (for example ’x’ for a node) for each airbag reference geometry in model m:

```
var a = ReferenceGeometry.GetAll(m, ’foo’);
```

GetComments()

Description

Extracts the comments associated to a airbag reference geometry.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the airbag reference geometry a:

```
var comm_array = a.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of ReferenceGeometry objects for all of the flagged airbag reference geometrys in a model in PRIMER If the optional property argument is not given then an array of ReferenceGeometry objects is returned. If the property argument is given, then that property value for each airbag reference geometry is returned in the array instead of a ReferenceGeometry object

Arguments

- **Model** (Model)

Model to get airbag reference geometrys from

- **flag** (Flag)

Flag set on the airbag reference geometrys that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged airbag reference geometrys in the model

Returns

Array of ReferenceGeometry objects or properties

Return type

Array

Example

To make an array of ReferenceGeometry objects for all of the airbag reference geometrys in model m flagged with f:

```
var a = ReferenceGeometry.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the airbag reference geometrys in model m flagged with f:

```
var a = ReferenceGeometry.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the ReferenceGeometry object for a airbag reference geometry ID.

Arguments

- **Model** (Model)

Model to find the airbag reference geometry in

- **number** (integer)

number of the airbag reference geometry you want the ReferenceGeometry object for

Returns

ReferenceGeometry object (or null if airbag reference geometry does not exist).

Return type

ReferenceGeometry

Example

To get the ReferenceGeometry object for airbag reference geometry 100 in model m

```
var a = ReferenceGeometry.GetFromID(m, 100);
```

GetNode(nid*[integer]*)

Description

Returns the reference geometry coordinates for the node

Arguments

- **nid** (integer)

Node ID

Returns

An array containing the three reference coordinates (or null if the node is not on the reference geometry)

Return type

Array

Example

To get the reference coordinates of node number nid on reference geometry a

```
var coords = a.GetNode(nid);
```

GetParameter(prop*[string]*)

Description

Checks if a ReferenceGeometry property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the ReferenceGeometry.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

airbag reference geometry property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if ReferenceGeometry property a.example is a parameter:

```
Options.property_parameter_names = true;
if (a.GetParameter(a.example) ) do_something...
Options.property_parameter_names = false;
```

To check if ReferenceGeometry property a.example is a parameter by using the GetParameter method:

```
if (a.ViewParameters().GetParameter(a.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this reference_geometry (*AIRBAG_REFERENCE_GEOMETRY). **Note that a carriage return is not added**. See also ReferenceGeometry.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for reference_geometry m:

```
var key = m.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the reference_geometry. **Note that a carriage return is not added**. See also ReferenceGeometry.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for airbag reference geometry a:

```
var cards = b.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last airbag reference geometry in the model.

Arguments

- **Model** (Model)

Model to get last airbag reference geometry in

Returns

ReferenceGeometry object (or null if there are no airbag reference geometrys in the model).

Return type

ReferenceGeometry

Example

To get the last airbag reference geometry in model m:

```
var a = ReferenceGeometry.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free airbag reference geometry label in the model. Also see ReferenceGeometry.FirstFreeLabel(), ReferenceGeometry.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free airbag reference geometry label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

ReferenceGeometry label.

Return type

Number

Example

To get the last free airbag reference geometry label in model m:

```
var label = ReferenceGeometry.LastFreeLabel(m);
```

Next()

Description

Returns the next airbag reference geometry in the model.

Arguments

No arguments

Returns

ReferenceGeometry object (or null if there are no more airbag reference geometrys in the model).

Return type

ReferenceGeometry

Example

To get the airbag reference geometry in model m after airbag reference geometry a:

```
var a = a.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) airbag reference geometry label in the model. Also see ReferenceGeometry.FirstFreeLabel(), ReferenceGeometry.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free airbag reference geometry label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

ReferenceGeometry label.

Return type

Number

Example

To get the next free airbag reference geometry label in model m:

```
var label = ReferenceGeometry.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a airbag reference geometry.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only airbag reference geometrys from that model can be picked. If the argument is a Flag then only airbag reference geometrys that are flagged with *limit* can be selected. If omitted, or null, any airbag reference geometrys from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

ReferenceGeometry object (or null if not picked)

Return type

ReferenceGeometry

Example

To pick a airbag reference geometry from model m giving the prompt 'Pick airbag reference geometry from screen':

```
var a = ReferenceGeometry.Pick('Pick airbag reference geometry from screen', m);
```

Previous()

Description

Returns the previous airbag reference geometry in the model.

Arguments

No arguments

Returns

ReferenceGeometry object (or null if there are no more airbag reference geometrys in the model).

Return type

ReferenceGeometry

Example

To get the airbag reference geometry in model m before airbag reference geometry a:

```
var a = a.Previous();
```

RemoveNode(nid*[integer]*)

Description

Removes a node from the reference geometry if it is on it

Arguments

- **nid** (integer)

Node ID

Returns

No return value.

Example

To remove node 11 from reference geometry a:

```
a.RemoveNode(11);
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the airbag reference geometrys in the model.

Arguments

- **Model** (Model)

Model that all airbag reference geometrys will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the airbag reference geometrys in model m, from 1000000:

```
ReferenceGeometry.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged airbag reference geometrys in the model.

Arguments

- **Model** (Model)

Model that all the flagged airbag reference geometrys will be renumbered in

- **flag** (Flag)

Flag set on the airbag reference geometrys that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the airbag reference geometrys in model m flagged with f, from 1000000:

```
ReferenceGeometry.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select airbag reference geometrys using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting airbag reference geometrys

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only airbag reference geometrys from that model can be selected. If the argument is a Flag then only airbag reference geometrys that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any airbag reference geometrys can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of airbag reference geometrys selected or null if menu cancelled

Return type

Number

Example

To select airbag reference geometrys from model m, flagging those selected with flag f, giving the prompt 'Select airbag reference geometrys':

```
ReferenceGeometry.Select(f, 'Select airbag reference geometrys', m);
```

To select airbag reference geometrys, flagging those selected with flag f but limiting selection to airbag reference geometrys flagged with flag l, giving the prompt 'Select airbag reference geometrys':

```
ReferenceGeometry.Select(f, 'Select airbag reference geometrys', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the airbag reference geometry.

Arguments

- **flag** (Flag)

Flag to set on the airbag reference geometry

Returns

No return value

Example

To set flag f for airbag reference geometry a:

```
a.SetFlag(f);
```

SetNode(nid*[integer]*, x*[real]*, y*[real]*, z*[real]*)

Description

Adds a node to the reference geometry if not already there, otherwise just changes the coordinates

Arguments

- **nid** (integer)

Node ID

- **x** (real)

X reference coordinate

- **y** (real)

Y reference coordinate

- **z** (real)

Z reference coordinate

Returns

No return value.

Example

To add node 11 to reference geometry a with coordinates 12.0, 13.0, 14.0

```
a.SetNode(11, 12.0, 13.0, 14.0);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the airbag reference geometry. The airbag reference geometry will be sketched until you either call ReferenceGeometry.Unsketch(), ReferenceGeometry.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the airbag reference geometry is sketched. If omitted redraw is true. If you want to sketch several airbag reference geometrys and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch airbag reference geometry a:

```
a.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged airbag reference geometrys in the model. The airbag reference geometrys will be sketched until you either call ReferenceGeometry.Unsketch(), ReferenceGeometry.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged airbag reference geometrys will be sketched in

- **flag** (Flag)

Flag set on the airbag reference geometrys that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the airbag reference geometrys are sketched. If omitted redraw is true. If you want to sketch flagged airbag reference geometrys several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all airbag reference geometrys flagged with flag in model m:

```
ReferenceGeometry.SketchFlagged(m, flag);
```

Spool()

Description

Spools a reference geometry, entry by entry. See also ReferenceGeometry.StartSpool

Arguments

No arguments

Returns

An array containing the node ID and the three coordinates. Returns 0 if no more items

Return type

Array

Example

To spool reference geometry a:

```
var array;
a.StartSpool();
while (array = a.Spool())
{
    do something...
}
```

StartSpool()

Description

Starts a reference geometry spooling operation. See also ReferenceGeometry.Spool

Arguments

No arguments

Returns

No return value

Example

To start spooling reference geometry a:

```
a.StartSpool();
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of airbag reference geometrys in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing airbag reference geometrys should be counted. If false or omitted referenced but undefined airbag reference geometrys will also be included in the total.

Returns

number of airbag reference geometrys

Return type

Number

Example

To get the total number of airbag reference geometrys in model m:

```
var total = ReferenceGeometry.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the airbag reference geometrys in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all airbag reference geometrys will be unset in

- **flag** (Flag)

Flag to unset on the airbag reference geometrys

Returns

No return value

Example

To unset the flag f on all the airbag reference geometrys in model m:

```
ReferenceGeometry.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the airbag reference geometry.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the airbag reference geometry is unsketched. If omitted redraw is true. If you want to unsketch several airbag reference geometrys and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch airbag reference geometry a:

```
a.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all airbag reference geometrys.

Arguments

- **Model** (Model)

Model that all airbag reference geometrys will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the airbag reference geometrys are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all airbag reference geometrys in model m:

```
ReferenceGeometry.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged airbag reference geometrys in the model.

Arguments

- **Model** (Model)

Model that all airbag reference geometrys will be unsketched in

- **flag** (Flag)

Flag set on the airbag reference geometrys that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the airbag reference geometrys are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all airbag reference geometrys flagged with flag in model m:

```
ReferenceGeometry.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

ReferenceGeometry object.

Return type

ReferenceGeometry

Example

To check if ReferenceGeometry property a.example is a parameter by using the ReferenceGeometry.GetParameter() method:

```
if (a.ViewParameters().GetParameter(a.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for airbag reference geometry with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The airbag reference geometry object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for airbag reference geometry a:

```
a.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for airbag reference geometry a:

```
a.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for airbag reference geometry. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for airbag reference geometry a:

```
a.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this airbag reference geometry.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for airbag reference geometry a:

```
var xrefs = a.Xrefs();
```

---

toString()

Description

Creates a string containing the ReferenceGeometry data in keyword format. Note that this contains the keyword header and the keyword cards. See also ReferenceGeometry.Keyword() and ReferenceGeometry.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for ReferenceGeometry rdt in keyword format

```
var s = rdt.toString();
```

ShellReferenceGeometry class

The ShellReferenceGeometry class gives you access to airbag shell reference geometry cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- GetShell(eid*[integer]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- RemoveShell(eid*[integer]*)
- SetFlag(flag*[Flag]*)
- SetShell(eid*[integer]*, n1*[integer]*, n2*[integer]*, n3*[integer]*, n4*[integer]*, pid (optional)*[integer]*)
- Sketch(redraw (optional)*[boolean]*)
- Spool()
- StartSpool()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

ShellReferenceGeometry properties

Name	Type	Description
aid	integer	ShellReferenceGeometry number. Also see the label property which is an alternative name for this.
exists (read only)	logical	true if airbag shell reference geometry exists, false if referred to but not defined.
id	logical	Turns _ID on or OFF
include	integer	The Include file number that the airbag shell reference geometry is in.
iout	integer	Flag for outputting the current reference node coordinates
label	integer	ShellReferenceGeometry number. Also see the aid property which is an alternative name for this.
model (read only)	integer	The Model number that the airbag shell reference geometry is in.
nid	integer	Node number for origin
rdt	logical	Turns _RDT on or OFF
sx	real	Scale factor in X direction
sy	real	Scale factor in Y direction
sz	real	Scale factor in Z direction

Detailed Description

The ShellReferenceGeometry class allows you to create, modify, edit and manipulate airbag shell reference geometry cards. See the documentation below for more details.

Constructor

new ShellReferenceGeometry(Model*[Model]*, aid (optional)*[integer]*)

Description

Create a new ShellReferenceGeometry object.

Arguments

- **Model** (Model)

Model that ShellReferenceGeometry will be created in

- **aid (optional)** (integer)

ShellReferenceGeometry number to set _ID suffix

Returns

ShellReferenceGeometry object

Return type

ShellReferenceGeometry

Example

To create a new ShellReferenceGeometry in model m

```
var a = new ShellReferenceGeometry(m);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a airbag shell reference geometry.

Arguments

- **Comment** (Comment)

Comment that will be attached to the airbag shell reference geometry

Returns

No return value

Example

To associate comment c to the airbag shell reference geometry a:

```
a.AssociateComment(c);
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse airbag shell reference geometry a:

```
a.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the airbag shell reference geometry.

Arguments

- **flag** (Flag)

Flag to clear on the airbag shell reference geometry

Returns

No return value

Example

To clear flag f for airbag shell reference geometry a:

```
a.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the airbag shell reference geometry. The target include of the copied airbag shell reference geometry can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

ShellReferenceGeometry object

Return type

ShellReferenceGeometry

Example

To copy airbag shell reference geometry a into airbag shell reference geometry z:

```
var z = a.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a airbag shell reference geometry

Arguments

- **Model** (Model)

Model that the airbag shell reference geometry will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

ShellReferenceGeometry object (or null if not made).

Return type

ShellReferenceGeometry

Example

To start creating a airbag shell reference geometry in model m:

```
var a = ShellReferenceGeometry.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a airbag shell reference geometry.

Arguments

- **Comment** (Comment)

Comment that will be detached from the airbag shell reference geometry

Returns

No return value

Example

To detach comment c from the airbag shell reference geometry a:

```
a.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit airbag shell reference geometry a:

```
a.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for airbag shell reference geometry with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description

autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The airbag shell reference geometry object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for airbag shell reference geometry a:

```
a.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for airbag shell reference geometry a:

```
a.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for airbag shell reference geometry. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for airbag shell reference geometry a:

```
a.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first airbag shell reference geometry in the model.

Arguments

- **Model** (Model)

Model to get first airbag shell reference geometry in

Returns

ShellReferenceGeometry object (or null if there are no airbag shell reference geometrys in the model).

Return type

ShellReferenceGeometry

Example

To get the first airbag shell reference geometry in model m:

```
var a = ShellReferenceGeometry.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free airbag shell reference geometry label in the model. Also see ShellReferenceGeometry.LastFreeLabel(), ShellReferenceGeometry.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free airbag shell reference geometry label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

ShellReferenceGeometry label.

Return type

Number

Example

To get the first free airbag shell reference geometry label in model m:

```
var label = ShellReferenceGeometry.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the airbag shell reference geometrys in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all airbag shell reference geometrys will be flagged in

- **flag** (Flag)

Flag to set on the airbag shell reference geometrys

Returns

No return value

Example

To flag all of the airbag shell reference geometrys with flag f in model m:

```
ShellReferenceGeometry.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the airbag shell reference geometry is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the airbag shell reference geometry

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if airbag shell reference geometry a has flag f set on it:

```
if (a.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each airbag shell reference geometry in the model.
Note that ForEach has been designed to make looping over airbag shell reference geometrys as fast as possible and so has some limitations.
Firstly, a single temporary ShellReferenceGeometry object is created and on each function call it is updated with the current airbag shell reference geometry data. This means that you should not try to store the ShellReferenceGeometry object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new airbag shell reference geometrys inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all airbag shell reference geometrys are in

- **func** (function)

Function to call for each airbag shell reference geometry

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the airbag shell reference geometrys in model m:

```
ShellReferenceGeometry.ForEach(m, test);
function test(a)
{
// a is ShellReferenceGeometry object
}
```

To call function test for all of the airbag shell reference geometrys in model m with optional object:

```
var data = { x:0, y:0 };
ShellReferenceGeometry.ForEach(m, test, data);
function test(a, extra)
{
// a is ShellReferenceGeometry object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of ShellReferenceGeometry objects or properties for all of the airbag shell reference geometrys in a model in PRIMER. If the optional property argument is not given then an array of ShellReferenceGeometry objects is returned. If the property argument is given, that property value for each airbag shell reference geometry is returned in the array instead of a ShellReferenceGeometry object

Arguments

- **Model** (Model)

Model to get airbag shell reference geometrys from

- **property (optional)** (string)

Name for property to get for all airbag shell reference geometrys in the model

Returns

Array of ShellReferenceGeometry objects or properties

Return type

Array

Example

To make an array of ShellReferenceGeometry objects for all of the airbag shell reference geometrys in model m:

```
var a = ShellReferenceGeometry.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each airbag shell reference geometry in model m:

```
var a = ShellReferenceGeometry.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a airbag shell reference geometry.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the airbag shell reference geometry a:

```
var comm_array = a.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of ShellReferenceGeometry objects for all of the flagged airbag shell reference geometrys in a model in PRIMER If the optional property argument is not given then an array of ShellReferenceGeometry objects is returned. If the property argument is given, then that property value for each airbag shell reference geometry is returned in the array instead of a ShellReferenceGeometry object

Arguments

- **Model** (Model)

Model to get airbag shell reference geometrys from

- **flag** (Flag)

Flag set on the airbag shell reference geometrys that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged airbag shell reference geometrys in the model

Returns

Array of ShellReferenceGeometry objects or properties

Return type

Array

Example

To make an array of ShellReferenceGeometry objects for all of the airbag shell reference geometrys in model m flagged with f:

```
var a = ShellReferenceGeometry.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the airbag shell reference geometrys in model m flagged with f:

```
var a = ShellReferenceGeometry.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the ShellReferenceGeometry object for a airbag shell reference geometry ID.

Arguments

- **Model** (Model)

Model to find the airbag shell reference geometry in

- **number** (integer)

number of the airbag shell reference geometry you want the ShellReferenceGeometry object for

Returns

ShellReferenceGeometry object (or null if airbag shell reference geometry does not exist).

Return type

ShellReferenceGeometry

Example

To get the ShellReferenceGeometry object for airbag shell reference geometry 100 in model m

```
var a = ShellReferenceGeometry.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a ShellReferenceGeometry property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the ShellReferenceGeometry.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

airbag shell reference geometry property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if ShellReferenceGeometry property a.example is a parameter:

```
Options.property_parameter_names = true;
if (a.GetParameter(a.example) ) do_something...
Options.property_parameter_names = false;
```

To check if ShellReferenceGeometry property a.example is a parameter by using the GetParameter method:

```
if (a.ViewParameters().GetParameter(a.example) ) do_something...
```

GetShell(eid*[integer]*)

Description

Returns the shell reference geometry nodes and pid for the shell

Arguments

- **eid** (integer)

Shell element ID

Returns

An array containing the four reference node labels and the part ID (or null if the shell is not on the shell reference geometry)

Return type

Array

Example

To get the node and part data of shell number eid on shell reference geometry a

```
var data = a.GetShell(eid);
var n1 = data[0];
var n2 = data[1];
var n3 = data[2];
var n4 = data[3];
var pid = data[4];
```

Keyword()

Description

Returns the keyword for this shell_reference_geometry (*AIRBAG_SHELL_REFERENCE_GEOMETRY). **Note that a carriage return is not added**. See also ShellReferenceGeometry.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for shell_reference_geometry a:

```
var key = a.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the shell_reference_geometry. **Note that a carriage return is not added**. See also ShellReferenceGeometry.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for airbag shell reference geometry a:

```
var cards = b.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last airbag shell reference geometry in the model.

Arguments

- **Model** (Model)

Model to get last airbag shell reference geometry in

Returns

ShellReferenceGeometry object (or null if there are no airbag shell reference geometrys in the model).

Return type

ShellReferenceGeometry

Example

To get the last airbag shell reference geometry in model m:

```
var a = ShellReferenceGeometry.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free airbag shell reference geometry label in the model. Also see ShellReferenceGeometry.FirstFreeLabel(), ShellReferenceGeometry.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free airbag shell reference geometry label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

ShellReferenceGeometry label.

Return type

Number

Example

To get the last free airbag shell reference geometry label in model m:

```
var label = ShellReferenceGeometry.LastFreeLabel(m);
```

Next()

Description

Returns the next airbag shell reference geometry in the model.

Arguments

No arguments

Returns

ShellReferenceGeometry object (or null if there are no more airbag shell reference geometrys in the model).

Return type

ShellReferenceGeometry

Example

To get the airbag shell reference geometry in model m after airbag shell reference geometry a:

```
var a = a.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) airbag shell reference geometry label in the model. Also see ShellReferenceGeometry.FirstFreeLabel(), ShellReferenceGeometry.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free airbag shell reference geometry label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

ShellReferenceGeometry label.

Return type

Number

Example

To get the next free airbag shell reference geometry label in model m:

```
var label = ShellReferenceGeometry.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a airbag shell reference geometry.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only airbag shell reference geometrys from that model can be picked. If the argument is a Flag then only airbag shell reference geometrys that are flagged with *limit* can be selected. If omitted, or null, any airbag shell reference geometrys from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

ShellReferenceGeometry object (or null if not picked)

Return type

ShellReferenceGeometry

Example

To pick a airbag shell reference geometry from model m giving the prompt 'Pick airbag shell reference geometry from screen':

```
var a = ShellReferenceGeometry.Pick('Pick airbag shell reference geometry from screen', m);
```

Previous()

Description

Returns the previous airbag shell reference geometry in the model.

Arguments

No arguments

Returns

ShellReferenceGeometry object (or null if there are no more airbag shell reference geometrys in the model).

Return type

ShellReferenceGeometry

Example

To get the airbag shell reference geometry in model m before airbag shell reference geometry a:

```
var a = a.Previous();
```

RemoveShell(eid*[integer]*)

Description

Removes a shell from the shell reference geometry if it is on it

Arguments

- **eid** (integer)

Element ID

Returns

No return value.

Example

To remove shell 11 from shell reference geometry a:

```
a.RemoveShell(11);
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the airbag shell reference geometrys in the model.

Arguments

- **Model** (Model)

Model that all airbag shell reference geometrys will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the airbag shell reference geometrys in model m, from 1000000:

```
ShellReferenceGeometry.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged airbag shell reference geometrys in the model.

Arguments

- **Model** (Model)

Model that all the flagged airbag shell reference geometrys will be renumbered in

- **flag** (Flag)

Flag set on the airbag shell reference geometrys that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the airbag shell reference geometrys in model m flagged with f, from 1000000:

```
ShellReferenceGeometry.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select airbag shell reference geometrys using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting airbag shell reference geometrys

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only airbag shell reference geometrys from that model can be selected. If the argument is a Flag then only airbag shell reference geometrys that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any airbag shell reference geometrys can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of airbag shell reference geometrys selected or null if menu cancelled

Return type

Number

Example

To select airbag shell reference geometrys from model m, flagging those selected with flag f, giving the prompt 'Select airbag shell reference geometrys':

```
ShellReferenceGeometry.Select(f, 'Select airbag shell reference geometrys', m);
```

To select airbag shell reference geometrys, flagging those selected with flag f but limiting selection to airbag shell reference geometrys flagged with flag l, giving the prompt 'Select airbag shell reference geometrys':

```
ShellReferenceGeometry.Select(f, 'Select airbag shell reference geometrys', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the airbag shell reference geometry.

Arguments

- **flag** (Flag)

Flag to set on the airbag shell reference geometry

Returns

No return value

Example

To set flag f for airbag shell reference geometry a:

```
a.SetFlag(f);
```

SetShell(eid*[integer]*, n1*[integer]*, n2*[integer]*, n3*[integer]*, n4*[integer]*, pid (optional)*[integer]*)

Description

Adds a shell to the shell reference geometry if not already there, otherwise just changes the reference nodes

Arguments

- **eid** (integer)

Element ID

- **n1** (integer)

Nodal point 1

- **n2** (integer)

Nodal point 2

- **n3** (integer)

Nodal point 3

- **n4** (integer)

Nodal point 4

- **pid (optional)** (integer)

Part ID (ignored by Ansys LS-DYNA). If omitted pid will be zero.

Returns

No return value.

Example

To add shell 11 to shell reference geometry a with nodal points 12, 13, 14, 15 (and part ID 0):

```
a.SetShell(11, 12, 13, 14, 15);
```

To add shell 11 to shell reference geometry a with nodal points 12, 13, 14, 15 and pid 100:

```
a.SetShell(11, 12, 13, 14, 15, 100);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the airbag shell reference geometry. The airbag shell reference geometry will be sketched until you either call ShellReferenceGeometry.Unsketch(), ShellReferenceGeometry.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the airbag shell reference geometry is sketched. If omitted redraw is true. If you want to sketch several airbag shell reference geometrys and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch airbag shell reference geometry a:

```
a.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged airbag shell reference geometrys in the model. The airbag shell reference geometrys will be sketched until you either call ShellReferenceGeometry.Unsketch(), ShellReferenceGeometry.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged airbag shell reference geometrys will be sketched in

- **flag** (Flag)

Flag set on the airbag shell reference geometrys that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the airbag shell reference geometrys are sketched. If omitted redraw is true. If you want to sketch flagged airbag shell reference geometrys several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all airbag shell reference geometrys flagged with flag in model m:

```
ShellReferenceGeometry.SketchFlagged(m, flag);
```

Spool()

Description

Spools a shell reference geometry, entry by entry. See also ShellReferenceGeometry.StartSpool

Arguments

No arguments

Returns

Returns an array containing the shell ID and the four nodal point labels. Returns 0 if no more items

Return type

Array

Example

To spool shell reference geometry a:

```
var array;
a.StartSpool();
while (array = a.Spool())
{
    do something...
}
```

StartSpool()

Description

Starts a shell reference geometry spooling operation. See also ShellReferenceGeometry.Spool

Arguments

No arguments

Returns

No return value

Example

To start spooling shell reference geometry a:

```
a.StartSpool();
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of airbag shell reference geometrys in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing airbag shell reference geometrys should be counted. If false or omitted referenced but undefined airbag shell reference geometrys will also be included in the total.

Returns

number of airbag shell reference geometrys

Return type

Number

Example

To get the total number of airbag shell reference geometrys in model m:

```
var total = ShellReferenceGeometry.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the airbag shell reference geometrys in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all airbag shell reference geometrys will be unset in

- **flag** (Flag)

Flag to unset on the airbag shell reference geometrys

Returns

No return value

Example

To unset the flag f on all the airbag shell reference geometrys in model m:

```
ShellReferenceGeometry.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the airbag shell reference geometry.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the airbag shell reference geometry is unsketched. If omitted redraw is true. If you want to unsketch several airbag shell reference geometrys and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch airbag shell reference geometry a:

```
a.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all airbag shell reference geometrys.

Arguments

- **Model** (Model)

Model that all airbag shell reference geometrys will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the airbag shell reference geometrys are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all airbag shell reference geometrys in model m:

```
ShellReferenceGeometry.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged airbag shell reference geometrys in the model.

Arguments

- **Model** (Model)

Model that all airbag shell reference geometrys will be unsketched in

- **flag** (Flag)

Flag set on the airbag shell reference geometrys that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the airbag shell reference geometrys are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all airbag shell reference geometrys flagged with flag in model m:

```
ShellReferenceGeometry.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

ShellReferenceGeometry object.

Return type

ShellReferenceGeometry

Example

To check if ShellReferenceGeometry property a.example is a parameter by using the ShellReferenceGeometry.GetParameter() method:

```
if (a.ViewParameters().GetParameter(a.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for airbag shell reference geometry with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The airbag shell reference geometry object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for airbag shell reference geometry a:

```
a.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for airbag shell reference geometry a:

```
a.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for airbag shell reference geometry. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for airbag shell reference geometry a:

```
a.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this airbag shell reference geometry.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for airbag shell reference geometry a:

```
var xrefs = a.Xrefs();
```

---

toString()

Description

Creates a string containing the ShellReferenceGeometry data in keyword format. Note that this contains the keyword header and the keyword cards. See also ShellReferenceGeometry.Keyword() and ShellReferenceGeometry.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for ShellReferenceGeometry rdt in keyword format

```
var s = rdt.toString();
```

PrescribedAccelerometerRigid class

The PrescribedAccelerometerRigid class gives you access to define *BOUNDARY_PRESCRIBED_ACCELEROMETER_RIGID cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- GetRow(row*[integer]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- RemoveRow(row*[integer]*)
- SetFlag(flag*[Flag]*)
- SetRow(row*[integer]*, data*[Array of data]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

PrescribedAccelerometerRigid properties

Name	Type	Description
exists (read only)	logical	true if prescribed accelerometer rigid exists, false if referred to but not defined.
include	integer	The Include file number that the prescribed accelerometer rigid is in.
model (read only)	integer	The Model number that the prescribed accelerometer rigid is in.

nrow (read only)	integer	Number of accelerometer cards.
pid	integer	Part ID for rigid body whose motion is prescribed.
solv	integer	Solver type: 1 for Gaussian elimination or 2 for linear regression.

Detailed Description

The PrescribedAccelerometerRigid class allows you to create, modify, edit and manipulate boundary prescribed accelerometer rigid cards. See the documentation below for more details.

Constructor

new PrescribedAccelerometerRigid(Model*[Model]*, pid*[integer]*, solv (optional)*[integer]*)

Description

Create a new PrescribedAccelerometerRigid object.

Arguments

- **Model** (Model)

Model that prescribed accelerometer rigid will be created in

- **pid** (integer)

Part ID for rigid body whose motion is prescribed.

- **solv (optional)** (integer)

Solver type

Returns

PrescribedAccelerometerRigid object

Return type

PrescribedAccelerometerRigid

Example

To create a new prescribed accelerometer rigid in model m with part ID 10 and solver type 2 (linear regression):

```
var par = new PrescribedAccelerometerRigid(m, 10, 2);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a prescribed accelerometer rigid.

Arguments

- **Comment** (Comment)

Comment that will be attached to the prescribed accelerometer rigid

Returns

No return value

Example

To associate comment c to the prescribed accelerometer rigid par:

```
par.AssociateComment(c);
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse prescribed accelerometer rigid par:

```
par.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the prescribed accelerometer rigid.

Arguments

- **flag** (Flag)

Flag to clear on the prescribed accelerometer rigid

Returns

No return value

Example

To clear flag f for prescribed accelerometer rigid par:

```
par.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the prescribed accelerometer rigid. The target include of the copied prescribed accelerometer rigid can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

PrescribedAccelerometerRigid object

Return type

PrescribedAccelerometerRigid

Example

To copy prescribed accelerometer rigid par into prescribed accelerometer rigid z:

```
var z = par.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a prescribed accelerometer rigid

Arguments

- **Model** (Model)

Model that the prescribed accelerometer rigid will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

PrescribedAccelerometerRigid object (or null if not made).

Return type

PrescribedAccelerometerRigid

Example

To start creating a prescribed accelerometer rigid in model m:

```
var par = PrescribedAccelerometerRigid.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a prescribed accelerometer rigid.

Arguments

- **Comment** (Comment)

Comment that will be detached from the prescribed accelerometer rigid

Returns

No return value

Example

To detach comment c from the prescribed accelerometer rigid par:

```
par.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit prescribed accelerometer rigid par:

```
par.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for prescribed accelerometer rigid with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The prescribed accelerometer rigid object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for prescribed accelerometer rigid par:

```
par.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for prescribed accelerometer rigid par:

```
par.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for prescribed accelerometer rigid. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for prescribed accelerometer rigid par:

```
par.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first prescribed accelerometer rigid in the model.

Arguments

- **Model** (Model)

Model to get first prescribed accelerometer rigid in

Returns

PrescribedAccelerometerRigid object (or null if there are no prescribed accelerometer rigids in the model).

Return type

PrescribedAccelerometerRigid

Example

To get the first prescribed accelerometer rigid in model m:

```
var par = PrescribedAccelerometerRigid.First(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the prescribed accelerometer rigids in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all prescribed accelerometer rigids will be flagged in

- **flag** (Flag)

Flag to set on the prescribed accelerometer rigids

Returns

No return value

Example

To flag all of the prescribed accelerometer rigids with flag f in model m:

```
PrescribedAccelerometerRigid.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the prescribed accelerometer rigid is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the prescribed accelerometer rigid

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if prescribed accelerometer rigid par has flag f set on it:

```
if (par.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each prescribed accelerometer rigid in the model.
Note that ForEach has been designed to make looping over prescribed accelerometer rigids as fast as possible and so has some limitations.
Firstly, a single temporary PrescribedAccelerometerRigid object is created and on each function call it is updated with the current prescribed accelerometer rigid data. This means that you should not try to store the PrescribedAccelerometerRigid object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new prescribed accelerometer rigids inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all prescribed accelerometer rigids are in

- **func** (function)

Function to call for each prescribed accelerometer rigid

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the prescribed accelerometer rigids in model m:

```
PrescribedAccelerometerRigid.ForEach(m, test);
function test(par)
{
// par is PrescribedAccelerometerRigid object
}
```

To call function test for all of the prescribed accelerometer rigids in model m with optional object:

```
var data = { x:0, y:0 };
PrescribedAccelerometerRigid.ForEach(m, test, data);
function test(par, extra)
{
// par is PrescribedAccelerometerRigid object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of PrescribedAccelerometerRigid objects or properties for all of the prescribed accelerometer rigids in a model in PRIMER. If the optional property argument is not given then an array of PrescribedAccelerometerRigid objects is returned. If the property argument is given, that property value for each prescribed accelerometer rigid is returned in the array instead of a PrescribedAccelerometerRigid object

Arguments

- **Model** (Model)

Model to get prescribed accelerometer rigids from

- **property (optional)** (string)

Name for property to get for all prescribed accelerometer rigids in the model

Returns

Array of PrescribedAccelerometerRigid objects or properties

Return type

Array

Example

To make an array of PrescribedAccelerometerRigid objects for all of the prescribed accelerometer rigids in model m:

```
var a = PrescribedAccelerometerRigid.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each prescribed accelerometer rigid in model m:

```
var a = PrescribedAccelerometerRigid.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a prescribed accelerometer rigid.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the prescribed accelerometer rigid par:

```
var comm_array = par.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of PrescribedAccelerometerRigid objects for all of the flagged prescribed accelerometer rigids in a model in PRIMER If the optional property argument is not given then an array of PrescribedAccelerometerRigid objects is returned. If the property argument is given, then that property value for each prescribed accelerometer rigid is returned in the array instead of a PrescribedAccelerometerRigid object

Arguments

- **Model** (Model)

Model to get prescribed accelerometer rigids from

- **flag** (Flag)

Flag set on the prescribed accelerometer rigids that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged prescribed accelerometer rigids in the model

Returns

Array of PrescribedAccelerometerRigid objects or properties

Return type

Array

Example

To make an array of PrescribedAccelerometerRigid objects for all of the prescribed accelerometer rigids in model m flagged with f:

```
var par = PrescribedAccelerometerRigid.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the prescribed accelerometer rigids in model m flagged with f:

```
var a = PrescribedAccelerometerRigid.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the PrescribedAccelerometerRigid object for a prescribed accelerometer rigid ID.

Arguments

- **Model** (Model)

Model to find the prescribed accelerometer rigid in

- **number** (integer)

number of the prescribed accelerometer rigid you want the PrescribedAccelerometerRigid object for

Returns

PrescribedAccelerometerRigid object (or null if prescribed accelerometer rigid does not exist).

Return type

PrescribedAccelerometerRigid

Example

To get the PrescribedAccelerometerRigid object for prescribed accelerometer rigid 100 in model m

```
var par = PrescribedAccelerometerRigid.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a PrescribedAccelerometerRigid property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the PrescribedAccelerometerRigid.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

prescribed accelerometer rigid property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if PrescribedAccelerometerRigid property par.example is a parameter:

```
Options.property_parameter_names = true;
if (par.GetParameter(par.example) ) do_something...
Options.property_parameter_names = false;
```

To check if PrescribedAccelerometerRigid property par.example is a parameter by using the GetParameter method:

```
if (par.ViewParameters().GetParameter(par.example) ) do_something...
```

GetRow(row*[integer]*)

Description

Returns the data for a row in the prescribed accelerometer rigid.

Arguments

- **row** (integer)

The row you want the data for. **Note row indices start at 0**.

Returns

An array of numbers containing the row variables NID, CID, LCIDX, LCIDY and LCIDZ.

Return type

Number

Example

To get the data for the 2nd row in prescribed accelerometer rigid par:

```
var data = par.GetRow(1);
```

Keyword()

Description

Returns the keyword for this prescribed accelerometer rigid. **Note that a carriage return is not added**. See also PrescribedAccelerometerRigid.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for prescribed accelerometer rigid par:

```
var key = par.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the prescribed accelerometer rigid. **Note that a carriage return is not added**. See also PrescribedAccelerometerRigid.Keyword()

Arguments

No arguments

Returns
string containing the cards.

Return type
String

Example
To get the cards for prescribed accelerometer rigid par:
```
var cards = par.KeywordCards();
```

Last(Model[*Model*]) [static]

Description
Returns the last prescribed accelerometer rigid in the model.

Arguments
- **Model** (Model)

Model to get last prescribed accelerometer rigid in

Returns
PrescribedAccelerometerRigid object (or null if there are no prescribed accelerometer rigids in the model).

Return type
PrescribedAccelerometerRigid

Example
To get the last prescribed accelerometer rigid in model m:
```
var par = PrescribedAccelerometerRigid.Last(m);
```

Next()

Description
Returns the next prescribed accelerometer rigid in the model.

Arguments
No arguments

Returns
PrescribedAccelerometerRigid object (or null if there are no more prescribed accelerometer rigids in the model).

Return type
PrescribedAccelerometerRigid

Example
To get the prescribed accelerometer rigid in model m after prescribed accelerometer rigid par:
```
var par = par.Next();
```

Previous()

Description

Returns the previous prescribed accelerometer rigid in the model.

Arguments

No arguments

Returns

PrescribedAccelerometerRigid object (or null if there are no more prescribed accelerometer rigids in the model).

Return type

PrescribedAccelerometerRigid

Example

To get the prescribed accelerometer rigid in model m before prescribed accelerometer rigid par:

```
var par = par.Previous();
```

RemoveRow(row*[integer]*)

Description

Removes the data for a row in *BOUNDARY_PRESCRIBED_ACCELEROMETER_RIGID.

Arguments

- **row** (integer)

The row you want to remove the data for. **Note that row indices start at 0**.

Returns

No return value.

Example

To remove the second row of data for prescribed accelerometer rigid par:

```
par.RemoveRow(1);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select prescribed accelerometer rigids using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting prescribed accelerometer rigids

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only prescribed accelerometer rigids from that model can be selected. If the argument is a Flag then only prescribed accelerometer rigids that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any prescribed accelerometer rigids can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of prescribed accelerometer rigids selected or null if menu cancelled

Return type

Number

Example

To select prescribed accelerometer rigids from model m, flagging those selected with flag f, giving the prompt 'Select prescribed accelerometer rigids':

```
PrescribedAccelerometerRigid.Select(f, 'Select prescribed accelerometer rigids', m);
```

To select prescribed accelerometer rigids, flagging those selected with flag f but limiting selection to prescribed accelerometer rigids flagged with flag l, giving the prompt 'Select prescribed accelerometer rigids':

```
PrescribedAccelerometerRigid.Select(f, 'Select prescribed accelerometer rigids', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the prescribed accelerometer rigid.

Arguments

- **flag** (Flag)

Flag to set on the prescribed accelerometer rigid

Returns

No return value

Example

To set flag f for prescribed accelerometer rigid par:

```
par.SetFlag(f);
```

SetRow(row*[integer]*, data*[Array of data]*)

Description

Sets the data for a row in *BOUNDARY_PRESCRIBED_ACCELEROMETER_RIGID.

Arguments

- **row** (integer)

The row you want to set the data for. **Note that row indices start at 0**.

- **data** (Array of data)

The data you want to set the row to

Returns

No return value.

Example

To set the second row of data for prescribed accelerometer rigid par to be node 11, coordinate system 12, and load curves 13, 14, 15:

```
var array = [11, 12, 13, 14, 15];
par.SetRow(1, array);
```

To append a new row of data (using the same array of values):

```
par.SetRow(par.nrow, array);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of prescribed accelerometer rigids in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing prescribed accelerometer rigids should be counted. If false or omitted referenced but undefined prescribed accelerometer rigids will also be included in the total.

Returns

number of prescribed accelerometer rigids

Return type

Number

Example

To get the total number of prescribed accelerometer rigids in model m:

```
var total = PrescribedAccelerometerRigid.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the prescribed accelerometer rigids in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all prescribed accelerometer rigids will be unset in

- **flag** (Flag)

Flag to unset on the prescribed accelerometer rigids

Returns

No return value

Example

To unset the flag f on all the prescribed accelerometer rigids in model m:

```
PrescribedAccelerometerRigid.UnflagAll(m, f);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

PrescribedAccelerometerRigid object.

Return type

PrescribedAccelerometerRigid

Example

To check if PrescribedAccelerometerRigid property par.example is a parameter by using the PrescribedAccelerometerRigid.GetParameter() method:

```
if (par.ViewParameters().GetParameter(par.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for prescribed accelerometer rigid with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The prescribed accelerometer rigid object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for prescribed accelerometer rigid par:

```
par.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for prescribed accelerometer rigid par:

```
par.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for prescribed accelerometer rigid. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for prescribed accelerometer rigid par:

```
par.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this prescribed accelerometer rigid.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for prescribed accelerometer rigid par:

```
var xrefs = par.Xrefs();
```

---

toString()

Description

Creates a string containing the prescribed accelerometer rigid data in keyword format. Note that this contains the keyword header and the keyword cards. See also PrescribedAccelerometerRigid.Keyword() and PrescribedAccelerometerRigid.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for prescribed accelerometer rigid par in keyword format

```
var s = par.toString();
```

PrescribedOrientationRigid class

The PrescribedOrientationRigid class gives you access to define *BOUNDARY_PRESCRIBED_ORIENTATION_RIGID cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

PrescribedOrientationRigid constants

Name	Description
PrescribedOrientationRigid.ANGLES	Boundary is *BOUNDARY_PRESCRIBED_ORIENTATION_RIGID_ANGLES.

PrescribedOrientationRigid.DIRCOS	Boundary is *BOUNDARY_PRESCRIBED_ORIENTATION_RIGID_DIRCOS.
PrescribedOrientationRigid.EULERP	Boundary is *BOUNDARY_PRESCRIBED_ORIENTATION_RIGID_EULERP.
PrescribedOrientationRigid.VECTOR	Boundary is *BOUNDARY_PRESCRIBED_ORIENTATION_RIGID_VECTOR.

PrescribedOrientationRigid properties

Name	Type	Description
birth	real	Time prior to which the body moves freely under the action of other agents.
body	integer	Reference axes: 0 for rotations about axes fixed in PIDA or 1 for those fixed in PIDB.
death	real	Time when the body is freed from the restriction.
exists (read only)	logical	true if prescribed orientation rigid exists, false if referred to but not defined.
heading	string	PrescribedOrientationRigid heading
id	logical	true if _ID option is set, false if not.
include	integer	The Include file number that the prescribed orientation rigid is in.
intrp	integer	Interpolation method: 1 for linear interpolation or 2 for cubic spline interpolation.
intrp	integer	Interpolation method: 1 for linear interpolation or 2 for cubic spline interpolation.
iseq	integer	Specifies the sequence in which the rotations are performed.
ishft	integer	Angle shift: 1 for unaltered angle curves or 2 for angle data shift in LCIDQi curves eliminating discontinuities.
lcidc11	integer	Load curve ID specifying direction cosine C11 as function of time.
lcidc12	integer	Load curve ID specifying direction cosine C12 as function of time.
lcidc13	integer	Load curve ID specifying direction cosine C13 as function of time.
lcidc21	integer	Load curve ID specifying direction cosine C21 as function of time.
lcidc22	integer	Load curve ID specifying direction cosine C22 as function of time.
lcidc23	integer	Load curve ID specifying direction cosine C23 as function of time.
lcidc31	integer	Load curve ID specifying direction cosine C31 as function of time.
lcidc32	integer	Load curve ID specifying direction cosine C32 as function of time.
lcidc33	integer	Load curve ID specifying direction cosine C33 as function of time.
lcide1	integer	Load curve ID specifying Euler parameter e1 as function of time.
lcide2	integer	Load curve ID specifying Euler parameter e2 as function of time.
lcide3	integer	Load curve ID specifying Euler parameter e3 as function of time.
lcide4	integer	Load curve ID specifying Euler parameter e4 as function of time.
lcidq1	integer	Load curve ID specifying orientation angle q1 as function of time.
lcidq2	integer	Load curve ID specifying orientation angle q2 as function of time.
lcidq3	integer	Load curve ID specifying orientation angle q3 as function of time.
lcids	integer	Load curve ID specifying spin speed of PIDB about axis parallel to vector.
lcidv1	integer	Load curve ID specifying vector measure number v1 as function of time.

lcidv2	integer	Load curve ID specifying vector measure number v2 as function of time.
lcidv3	integer	Load curve ID specifying vector measure number v3 as function of time.
model (read only)	integer	The Model number that the prescribed orientation rigid is in.
option	constant	The Boundary Prescribed Orientation Rigid option. Can be PrescribedOrientationRigid.DIRCOS, PrescribedOrientationRigid.ANGLES, PrescribedOrientationRigid.EULERP or PrescribedOrientationRigid.VECTOR.
pida	integer	Part ID for rigid body A.
pidb	integer	Part ID for rigid body B whose orientation is prescribed.
toffset	integer	Time offset flag.
valspin	real	Constant value for spin speed of PIDB about axis parallel to vector. Used when LCIDS is 0.

Detailed Description

The PrescribedOrientationRigid class allows you to create, modify, edit and manipulate boundary prescribed orientation rigid cards. See the documentation below for more details.

Constructor

new PrescribedOrientationRigid(Model*[Model]*, option*[constant]*, pidb*[integer]*, label (optional)*[integer]*, heading (optional)*[string]*)

Description

Create a new PrescribedOrientationRigid object.

Arguments

- **Model** (Model)

Model that prescribed orientation rigid will be created in

- **option** (constant)

Suffix for boundary prescribed orientation rigid. Can be PrescribedOrientationRigid.DIRCOS PrescribedOrientationRigid.ANGLES PrescribedOrientationRigid.EULERP PrescribedOrientationRigid.VECTOR

- **pidb** (integer)

Part ID for rigid body B whose orientation is prescribed.

- **label (optional)** (integer)

PrescribedOrientationRigid number

- **heading (optional)** (string)

Title for the PrescribedOrientationRigid

Returns

PrescribedOrientationRigid object

Return type

PrescribedOrientationRigid

Example

To create a new prescribed orientation rigid in model m with part ID 10 and suffix _DIRCOS:

```
var por = new PrescribedOrientationRigid(m, PrescribedOrientationRigid.DIRCOS,
10);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a prescribed orientation rigid.

Arguments

- **Comment** (Comment)

Comment that will be attached to the prescribed orientation rigid

Returns

No return value

Example

To associate comment c to the prescribed orientation rigid por:

```
por.AssociateComment(c);
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse prescribed orientation rigid por:

```
por.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the prescribed orientation rigid.

Arguments

- **flag** (Flag)

Flag to clear on the prescribed orientation rigid

Returns

No return value

Example

To clear flag f for prescribed orientation rigid por:

```
por.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the prescribed orientation rigid. The target include of the copied prescribed orientation rigid can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

PrescribedOrientationRigid object

Return type

PrescribedOrientationRigid

Example

To copy prescribed orientation rigid por into prescribed orientation rigid z:

```
var z = por.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a prescribed orientation rigid

Arguments

- **Model** (Model)

Model that the prescribed orientation rigid will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

PrescribedOrientationRigid object (or null if not made).

Return type

PrescribedOrientationRigid

Example

To start creating a prescribed orientation rigid in model m:

```
var por = PrescribedOrientationRigid.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a prescribed orientation rigid.

Arguments

- **Comment** (Comment)

Comment that will be detached from the prescribed orientation rigid

Returns

No return value

Example

To detach comment c from the prescribed orientation rigid por:

```
por.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit prescribed orientation rigid por:

```
por.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for prescribed orientation rigid with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description

autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The prescribed orientation rigid object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for prescribed orientation rigid por:

```
por.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for prescribed orientation rigid por:

```
por.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for prescribed orientation rigid. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for prescribed orientation rigid por:

```
por.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first prescribed orientation rigid in the model.

Arguments

- **Model** (Model)

Model to get first prescribed orientation rigid in

Returns

PrescribedOrientationRigid object (or null if there are no prescribed orientation rigids in the model).

Return type

PrescribedOrientationRigid

Example

To get the first prescribed orientation rigid in model m:

```
var por = PrescribedOrientationRigid.First(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the prescribed orientation rigids in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all prescribed orientation rigids will be flagged in

- **flag** (Flag)

Flag to set on the prescribed orientation rigids

Returns

No return value

Example

To flag all of the prescribed orientation rigids with flag f in model m:

```
PrescribedOrientationRigid.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the prescribed orientation rigid is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the prescribed orientation rigid

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if prescribed orientation rigid por has flag f set on it:

```
if (por.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each prescribed orientation rigid in the model.
Note that ForEach has been designed to make looping over prescribed orientation rigids as fast as possible and so has some limitations.
Firstly, a single temporary PrescribedOrientationRigid object is created and on each function call it is updated with the current prescribed orientation rigid data. This means that you should not try to store the PrescribedOrientationRigid object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new prescribed orientation rigids inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all prescribed orientation rigids are in

- **func** (function)

Function to call for each prescribed orientation rigid

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the prescribed orientation rigids in model m:

```
PrescribedOrientationRigid.ForEach(m, test);
function test(por)
{
// por is PrescribedOrientationRigid object
}
```

To call function test for all of the prescribed orientation rigids in model m with optional object:

```
var data = { x:0, y:0 };
PrescribedOrientationRigid.ForEach(m, test, data);
function test(por, extra)
{
// por is PrescribedOrientationRigid object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of PrescribedOrientationRigid objects or properties for all of the prescribed orientation rigids in a model in PRIMER. If the optional property argument is not given then an array of PrescribedOrientationRigid objects is returned. If the property argument is given, that property value for each prescribed orientation rigid is returned in the array instead of a PrescribedOrientationRigid object

Arguments

- **Model** (Model)

Model to get prescribed orientation rigids from

- **property (optional)** (string)

Name for property to get for all prescribed orientation rigids in the model

Returns

Array of PrescribedOrientationRigid objects or properties

Return type

Array

Example

To make an array of PrescribedOrientationRigid objects for all of the prescribed orientation rigids in model m:

```
var a = PrescribedOrientationRigid.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each prescribed orientation rigid in model m:

```
var a = PrescribedOrientationRigid.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a prescribed orientation rigid.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the prescribed orientation rigid por:

```
var comm_array = por.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of PrescribedOrientationRigid objects for all of the flagged prescribed orientation rigids in a model in PRIMER If the optional property argument is not given then an array of PrescribedOrientationRigid objects is returned. If the property argument is given, then that property value for each prescribed orientation rigid is returned in the array instead of a PrescribedOrientationRigid object

Arguments

- **Model** (Model)

Model to get prescribed orientation rigids from

- **flag** (Flag)

Flag set on the prescribed orientation rigids that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged prescribed orientation rigids in the model

Returns

Array of PrescribedOrientationRigid objects or properties

Return type

Array

Example

To make an array of PrescribedOrientationRigid objects for all of the prescribed orientation rigids in model m flagged with f:

```
var por = PrescribedOrientationRigid.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the prescribed orientation rigids in model m flagged with f:

```
var a = PrescribedOrientationRigid.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the PrescribedOrientationRigid object for a prescribed orientation rigid ID.

Arguments

- **Model** (Model)

Model to find the prescribed orientation rigid in

- **number** (integer)

number of the prescribed orientation rigid you want the PrescribedOrientationRigid object for

Returns

PrescribedOrientationRigid object (or null if prescribed orientation rigid does not exist).

Return type

PrescribedOrientationRigid

Example

To get the PrescribedOrientationRigid object for prescribed orientation rigid 100 in model m

```
var por = PrescribedOrientationRigid.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a PrescribedOrientationRigid property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the PrescribedOrientationRigid.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

prescribed orientation rigid property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if PrescribedOrientationRigid property por.example is a parameter:

```
Options.property_parameter_names = true;
if (por.GetParameter(por.example) ) do_something...
Options.property_parameter_names = false;
```

To check if PrescribedOrientationRigid property por.example is a parameter by using the GetParameter method:

```
if (por.ViewParameters().GetParameter(por.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this prescribed orientation rigid. **Note that a carriage return is not added**. See also PrescribedOrientationRigid.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for prescribed orientation rigid por:

```
var key = por.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the prescribed orientation rigid. **Note that a carriage return is not added**. See also PrescribedOrientationRigid.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for prescribed orientation rigid por:

```
var cards = por.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last prescribed orientation rigid in the model.

Arguments

- **Model** (Model)

Model to get last prescribed orientation rigid in

Returns

PrescribedOrientationRigid object (or null if there are no prescribed orientation rigids in the model).

Return type

PrescribedOrientationRigid

Example

To get the last prescribed orientation rigid in model m:

```
var por = PrescribedOrientationRigid.Last(m);
```

Next()

Description

Returns the next prescribed orientation rigid in the model.

Arguments

No arguments

Returns

PrescribedOrientationRigid object (or null if there are no more prescribed orientation rigids in the model).

Return type

PrescribedOrientationRigid

Example

To get the prescribed orientation rigid in model m after prescribed orientation rigid por:

```
var por = por.Next();
```

Previous()

Description

Returns the previous prescribed orientation rigid in the model.

Arguments

No arguments

Returns

PrescribedOrientationRigid object (or null if there are no more prescribed orientation rigids in the model).

Return type

PrescribedOrientationRigid

Example

To get the prescribed orientation rigid in model m before prescribed orientation rigid por:

```
var por = por.Previous();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select prescribed orientation rigids using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting prescribed orientation rigids

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only prescribed orientation rigids from that model can be selected. If the argument is a Flag then only prescribed orientation rigids that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any prescribed orientation rigids can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of prescribed orientation rigids selected or null if menu cancelled

Return type

Number

Example

To select prescribed orientation rigids from model m, flagging those selected with flag f, giving the prompt 'Select prescribed orientation rigids':

```
PrescribedOrientationRigid.Select(f, 'Select prescribed orientation rigids', m);
```

To select prescribed orientation rigids, flagging those selected with flag f but limiting selection to prescribed orientation rigids flagged with flag l, giving the prompt 'Select prescribed orientation rigids':

```
PrescribedOrientationRigid.Select(f, 'Select prescribed orientation rigids', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the prescribed orientation rigid.

Arguments

- **flag** (Flag)

Flag to set on the prescribed orientation rigid

Returns

No return value

Example

To set flag f for prescribed orientation rigid por:

```
por.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the prescribed orientation rigid. The prescribed orientation rigid will be sketched until you either call PrescribedOrientationRigid.Unsketch(), PrescribedOrientationRigid.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the prescribed orientation rigid is sketched. If omitted redraw is true. If you want to sketch several prescribed orientation rigids and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch prescribed orientation rigid por:

```
por.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged prescribed orientation rigids in the model. The prescribed orientation rigids will be sketched until you either call PrescribedOrientationRigid.Unsketch(), PrescribedOrientationRigid.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged prescribed orientation rigids will be sketched in

- **flag** (Flag)

Flag set on the prescribed orientation rigids that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the prescribed orientation rigids are sketched. If omitted redraw is true. If you want to sketch flagged prescribed orientation rigids several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all prescribed orientation rigids flagged with flag in model m:

```
PrescribedOrientationRigid.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of prescribed orientation rigids in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing prescribed orientation rigids should be counted. If false or omitted referenced but undefined prescribed orientation rigids will also be included in the total.

Returns

number of prescribed orientation rigids

Return type

Number

Example

To get the total number of prescribed orientation rigids in model m:

```
var total = PrescribedOrientationRigid.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the prescribed orientation rigids in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all prescribed orientation rigids will be unset in

- **flag** (Flag)

Flag to unset on the prescribed orientation rigids

Returns

No return value

Example

To unset the flag f on all the prescribed orientation rigids in model m:

```
PrescribedOrientationRigid.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the prescribed orientation rigid.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the prescribed orientation rigid is unsketched. If omitted redraw is true. If you want to unsketch several prescribed orientation rigids and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch prescribed orientation rigid por:

```
por.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all prescribed orientation rigids.

Arguments

- **Model** (Model)

Model that all prescribed orientation rigids will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the prescribed orientation rigids are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all prescribed orientation rigids in model m:

```
PrescribedOrientationRigid.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged prescribed orientation rigids in the model.

Arguments

- **Model** (Model)

Model that all prescribed orientation rigids will be unsketched in

- **flag** (Flag)

Flag set on the prescribed orientation rigids that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the prescribed orientation rigids are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all prescribed orientation rigids flagged with flag in model m:

```
PrescribedOrientationRigid.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

PrescribedOrientationRigid object.

Return type

PrescribedOrientationRigid

Example

To check if PrescribedOrientationRigid property por.example is a parameter by using the PrescribedOrientationRigid.GetParameter() method:

```
if (por.ViewParameters().GetParameter(por.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for prescribed orientation rigid with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description

autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The prescribed orientation rigid object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for prescribed orientation rigid por:

```
por.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for prescribed orientation rigid por:

```
por.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for prescribed orientation rigid. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for prescribed orientation rigid por:

```
por.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this prescribed orientation rigid.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for prescribed orientation rigid por:

```
var xrefs = por.Xrefs();
```

---

toString()

Description

Creates a string containing the prescribed orientation rigid data in keyword format. Note that this contains the keyword header and the keyword cards. See also PrescribedOrientationRigid.Keyword() and PrescribedOrientationRigid.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for prescribed orientation rigid por in keyword format

```
var s = por.toString();
```

PrescribedFinalGeometry class

The PrescribedFinalGeometry class gives you access to define boundary prescribed final_geometry cards in PRIMER.
More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetData(index*[integer]*)
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- RemoveData(index*[Integer]*)
- SetData(index*[Integer]*, nid*[integer]*, x*[real]*, y*[real]*, z*[real]*, lcid (optional)*[integer]*, death (optional)*[real]*, birth (optional)*[real]*)
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)

- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

PrescribedFinalGeometry properties

Name	Type	Description
bpfgid	integer	PrescribedFinalGeometry number. Also see the label property which is an alternative name for this.
deathd	real	Default death time.
exists (read only)	logical	true if boundary prescribed final geometry exists, false if referred to but not defined.
ibrth	integer	Flag to allow node input with death and birth times
id	integer	PrescribedFinalGeometry number. Also see the label property which is an alternative name for this.
include	integer	The Include file number that the boundary prescribed final geometry is in.
label	integer	PrescribedFinalGeometry number. Also see the bpfgid property which is an alternative name for this.
lcidf	integer	Default loadcurve number.
lines (read only)	integer	Number of lines of nodal data on the card.
model (read only)	integer	The Model number that the boundary prescribed final geometry is in.

Detailed Description

The PrescribedFinalGeometry class allows you to create, modify, edit and boundary prescribed final_geometry cards. See the documentation below for more details.

Constructor

new PrescribedFinalGeometry(Model*[Model]*, bpfgid*[PrescribedFinalGeometry]*)

Description

Create a new PrescribedFinalGeometry object.

Arguments

- **Model** (Model)

Model that PrescribedFinalGeometry will be created in

- **bpfgid** (PrescribedFinalGeometry)

PrescribedFinalGeometry number.

Returns

PrescribedFinalGeometry object

Return type

PrescribedFinalGeometry

Example

To create a new final geometry 99 in model m

```
var b = new PrescribedFinalGeometry(m, 99);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a boundary prescribed final geometry.

Arguments

- **Comment** (Comment)

Comment that will be attached to the boundary prescribed final geometry

Returns

No return value

Example

To associate comment c to the boundary prescribed final geometry b:

```
b.AssociateComment(c);
```

Blank()

Description

Blanks the boundary prescribed final geometry

Arguments

No arguments

Returns

No return value

Example

To blank boundary prescribed final geometry b:

```
b.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the boundary prescribed final geometrys in the model.

Arguments

- **Model** (Model)

Model that all boundary prescribed final geometrys will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the boundary prescribed final geometrys in model m:

```
PrescribedFinalGeometry.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged boundary prescribed final geometrys in the model.

Arguments

- **Model** (Model)

Model that all the flagged boundary prescribed final geometrys will be blanked in

- **flag** (Flag)

Flag set on the boundary prescribed final geometrys that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the boundary prescribed final geometrys in model m flagged with f:

```
PrescribedFinalGeometry.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the boundary prescribed final geometry is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if boundary prescribed final geometry b is blanked:

```
if (b.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse boundary prescribed final geometry b:

```
b.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the boundary prescribed final geometry.

Arguments

- **flag** (Flag)

Flag to clear on the boundary prescribed final geometry

Returns

No return value

Example

To clear flag f for boundary prescribed final geometry b:

```
b.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the boundary prescribed final geometry. The target include of the copied boundary prescribed final geometry can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

PrescribedFinalGeometry object

Return type

PrescribedFinalGeometry

Example

To copy boundary prescribed final geometry b into boundary prescribed final geometry z:

```
var z = b.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a boundary prescribed final geometry

Arguments

- **Model** (Model)

Model that the boundary prescribed final geometry will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

PrescribedFinalGeometry object (or null if not made).

Return type

PrescribedFinalGeometry

Example

To start creating a boundary prescribed final geometry in model m:

```
var b = PrescribedFinalGeometry.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a boundary prescribed final geometry.

Arguments

- **Comment** (Comment)

Comment that will be detached from the boundary prescribed final geometry

Returns

No return value

Example

To detach comment c from the boundary prescribed final geometry b:

```
b.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit boundary prescribed final geometry b:

```
b.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for boundary prescribed final geometry with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The boundary prescribed final geometry object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for boundary prescribed final geometry b:

```
b.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for boundary prescribed final geometry b:

```
b.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for boundary prescribed final geometry. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for boundary prescribed final geometry b:

```
b.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first boundary prescribed final geometry in the model.

Arguments

- **Model** (Model)

Model to get first boundary prescribed final geometry in

Returns

PrescribedFinalGeometry object (or null if there are no boundary prescribed final geometrys in the model).

Return type

PrescribedFinalGeometry

Example

To get the first boundary prescribed final geometry in model m:

```
var b = PrescribedFinalGeometry.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free boundary prescribed final geometry label in the model. Also see PrescribedFinalGeometry.LastFreeLabel(), PrescribedFinalGeometry.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free boundary prescribed final geometry label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

PrescribedFinalGeometry label.

Return type

Number

Example

To get the first free boundary prescribed final geometry label in model m:

```
var label = PrescribedFinalGeometry.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the boundary prescribed final geometrys in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all boundary prescribed final geometrys will be flagged in

- **flag** (Flag)

Flag to set on the boundary prescribed final geometrys

Returns

No return value

Example

To flag all of the boundary prescribed final geometrys with flag f in model m:

```
PrescribedFinalGeometry.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the boundary prescribed final geometry is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the boundary prescribed final geometry

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if boundary prescribed final geometry b has flag f set on it:

```
if (b.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each boundary prescribed final geometry in the model.
Note that ForEach has been designed to make looping over boundary prescribed final geometrys as fast as possible and so has some limitations.
Firstly, a single temporary PrescribedFinalGeometry object is created and on each function call it is updated with the current boundary prescribed final geometry data. This means that you should not try to store the PrescribedFinalGeometry object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new boundary prescribed final geometrys inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all boundary prescribed final geometrys are in

- **func** (function)

Function to call for each boundary prescribed final geometry

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the boundary prescribed final geometrys in model m:

```
PrescribedFinalGeometry.ForEach(m, test);
function test(b)
{
// b is PrescribedFinalGeometry object
}
```

To call function test for all of the boundary prescribed final geometrys in model m with optional object:

```
var data = { x:0, y:0 };
PrescribedFinalGeometry.ForEach(m, test, data);
function test(b, extra)
{
// b is PrescribedFinalGeometry object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of PrescribedFinalGeometry objects or properties for all of the boundary prescribed final geometrys in a model in PRIMER. If the optional property argument is not given then an array of PrescribedFinalGeometry objects is returned. If the property argument is given, that property value for each boundary prescribed final geometry is returned in the array instead of a PrescribedFinalGeometry object

Arguments

- **Model** (Model)

Model to get boundary prescribed final geometrys from

- **property (optional)** (string)

Name for property to get for all boundary prescribed final geometrys in the model

Returns

Array of PrescribedFinalGeometry objects or properties

Return type

Array

Example

To make an array of PrescribedFinalGeometry objects for all of the boundary prescribed final geometrys in model m:

```
var a = PrescribedFinalGeometry.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each boundary prescribed final geometry in model m:

```
var a = PrescribedFinalGeometry.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a boundary prescribed final geometry.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the boundary prescribed final geometry b:

```
var comm_array = b.GetComments();
```

GetData(index*[integer]*)

Description

Returns data for open-ended cards for a given row number in *BOUNDARY_PRESCRIBED_FINAL_GEOMETRY.

Arguments

- **index** (integer)

Index of open-ended card you want the data for. **Note that indices start at 0, not 1**.
0 <= index < lines

Returns

If IBRTH is **not** set, an array containing data (NID, X, Y, Z, LCID, DEATH). If IBRTH **is** set, an array containing data (NID, X, Y, Z, LCID, DEATH, BIRTH)

Return type

array

Example

To get the data for the 3rd open-ended row for boundary prescribed final geometry b:

```
var data = b.GetData(2);
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of PrescribedFinalGeometry objects for all of the flagged boundary prescribed final geometrys in a model in PRIMER If the optional property argument is not given then an array of PrescribedFinalGeometry objects is returned. If the property argument is given, then that property value for each boundary prescribed final geometry is returned in the array instead of a PrescribedFinalGeometry object

Arguments

- **Model** (Model)

Model to get boundary prescribed final geometrys from

- **flag** (Flag)

Flag set on the boundary prescribed final geometrys that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged boundary prescribed final geometrys in the model

Returns

Array of PrescribedFinalGeometry objects or properties

Return type

Array

Example

To make an array of PrescribedFinalGeometry objects for all of the boundary prescribed final geometrys in model m flagged with f:

```
var b = PrescribedFinalGeometry.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the boundary prescribed final geometrys in model m flagged with f:

```
var a = PrescribedFinalGeometry.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the PrescribedFinalGeometry object for a boundary prescribed final geometry ID.

Arguments

- **Model** (Model)

Model to find the boundary prescribed final geometry in

- **number** (integer)

number of the boundary prescribed final geometry you want the PrescribedFinalGeometry object for

Returns

PrescribedFinalGeometry object (or null if boundary prescribed final geometry does not exist).

Return type

PrescribedFinalGeometry

Example

To get the PrescribedFinalGeometry object for boundary prescribed final geometry 100 in model m

```
var b = PrescribedFinalGeometry.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a PrescribedFinalGeometry property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the PrescribedFinalGeometry.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

boundary prescribed final geometry property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if PrescribedFinalGeometry property b.example is a parameter:

```
Options.property_parameter_names = true;
if (b.GetParameter(b.example) ) do_something...
Options.property_parameter_names = false;
```

To check if PrescribedFinalGeometry property b.example is a parameter by using the GetParameter method:

```
if (b.ViewParameters().GetParameter(b.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this PrescribedFinalGeometry (*BOUNDARY_PRESCRIBED_FINAL_GEOMETRY). **Note that a carriage return is not added**. See also PrescribedFinalGeometry.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for PrescribedFinalGeometry bfg:

```
var key = bfg.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the PrescribedFinalGeometry. **Note that a carriage return is not added**. See also PrescribedFinalGeometry.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for PrescribedFinalGeometry bfg:

```
var cards = bfg.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last boundary prescribed final geometry in the model.

Arguments

- **Model** (Model)

Model to get last boundary prescribed final geometry in

Returns

PrescribedFinalGeometry object (or null if there are no boundary prescribed final geometrys in the model).

Return type

PrescribedFinalGeometry

Example

To get the last boundary prescribed final geometry in model m:

```
var b = PrescribedFinalGeometry.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free boundary prescribed final geometry label in the model. Also see PrescribedFinalGeometry.FirstFreeLabel(), PrescribedFinalGeometry.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free boundary prescribed final geometry label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

PrescribedFinalGeometry label.

Return type

Number

Example

To get the last free boundary prescribed final geometry label in model m:

```
var label = PrescribedFinalGeometry.LastFreeLabel(m);
```

Next()

Description

Returns the next boundary prescribed final geometry in the model.

Arguments

No arguments

Returns

PrescribedFinalGeometry object (or null if there are no more boundary prescribed final geometrys in the model).

Return type

PrescribedFinalGeometry

Example

To get the boundary prescribed final geometry in model m after boundary prescribed final geometry b:

```
var b = b.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) boundary prescribed final geometry label in the model. Also see PrescribedFinalGeometry.FirstFreeLabel(), PrescribedFinalGeometry.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free boundary prescribed final geometry label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

PrescribedFinalGeometry label.

Return type

Number

Example

To get the next free boundary prescribed final geometry label in model m:

```
var label = PrescribedFinalGeometry.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a boundary prescribed final geometry.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only boundary prescribed final geometrys from that model can be picked. If the argument is a Flag then only boundary prescribed final geometrys that are flagged with *limit* can be selected. If omitted, or null, any boundary prescribed final geometrys from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

PrescribedFinalGeometry object (or null if not picked)

Return type

PrescribedFinalGeometry

Example

To pick a boundary prescribed final geometry from model m giving the prompt 'Pick boundary prescribed final geometry from screen':

```
var b = PrescribedFinalGeometry.Pick('Pick boundary prescribed final geometry
from screen', m);
```

Previous()

Description

Returns the previous boundary prescribed final geometry in the model.

Arguments

No arguments

Returns

PrescribedFinalGeometry object (or null if there are no more boundary prescribed final geometrys in the model).

Return type

PrescribedFinalGeometry

Example

To get the boundary prescribed final geometry in model m before boundary prescribed final geometry b:

```
var b = b.Previous();
```

RemoveData(index*[Integer]*)

Description

Removes a line of data for a *BOUNDARY_PRESCRIBED_FINAL_GEOMETRY.

Arguments

- **index** (Integer)

The index of the *BOUNDARY_PRESCRIBED_FINAL_GEOMETRY data to remove. **Note that indices start at 0, not 1**.
0 <= index < lines

Returns

No return value.

Example

To remove row 2 (indices start with 0) of open-ended cards for *BOUNDARY_PRESCRIBED_FINAL_GEOMETRY b:

```
b.RemoveData(1);
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the boundary prescribed final geometrys in the model.

Arguments

- **Model** (Model)

Model that all boundary prescribed final geometrys will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the boundary prescribed final geometrys in model m, from 1000000:

```
PrescribedFinalGeometry.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged boundary prescribed final geometrys in the model.

Arguments

- **Model** (Model)

Model that all the flagged boundary prescribed final geometrys will be renumbered in

- **flag** (Flag)

Flag set on the boundary prescribed final geometrys that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the boundary prescribed final geometrys in model m flagged with f, from 1000000:

```
PrescribedFinalGeometry.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select boundary prescribed final geometrys using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting boundary prescribed final geometrys

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only boundary prescribed final geometrys from that model can be selected. If the argument is a Flag then only boundary prescribed final geometrys that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any boundary prescribed final geometrys can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of boundary prescribed final geometrys selected or null if menu cancelled

Return type

Number

Example

To select boundary prescribed final geometrys from model m, flagging those selected with flag f, giving the prompt 'Select boundary prescribed final geometrys':

```
PrescribedFinalGeometry.Select(f, 'Select boundary prescribed final geometrys', m);
```

To select boundary prescribed final geometrys, flagging those selected with flag f but limiting selection to boundary prescribed final geometrys flagged with flag l, giving the prompt 'Select boundary prescribed final geometrys':

```
PrescribedFinalGeometry.Select(f, 'Select boundary prescribed final geometrys', l);
```

SetData(index*[Integer]*, nid*[integer]*, x*[real]*, y*[real]*, z*[real]*, lcid (optional)*[integer]*, death (optional)*[real]*, birth (optional)*[real]*)

Description

Sets a line of data for a *BOUNDARY_PRESCRIBED_FINAL_GEOMETRY.

Arguments

- **index** (Integer)

The index of the *BOUNDARY_PRESCRIBED_FINAL_GEOMETRY data to set. **Note that indices start at 0, not 1**. 0 <= index <= lines

- **nid** (integer)

Node or negative node set number.

- **x** (real)

X coordinates of final geometry.

- **y** (real)

Y coordinates of final geometry.

- **z** (real)

Z coordinates of final geometry.

- **lcid (optional)** (integer)

Loadcurve number.

- **death (optional)** (real)

Death time.

- **birth (optional)** (real)

Birth time

Returns

No return value.

Example

To set values for row 2 (indices start with 0) of open-ended cards for *BOUNDARY_PRESCRIBED_FINAL_GEOMETRY b with the following specification: nid, x, y, z, lcid, death are 99, 0.1, 0.2, 0.3, 88, 100.0 respectively

```
b.SetData(1, 99, 0.1, 0.2, 0.3, 88, 100.0);
```

To append a new line of data (using the same example values):

```
b.SetData(b.lines, 99, 0.1, 0.2, 0.3, 88, 100.0);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the boundary prescribed final geometry.

Arguments

- **flag** (Flag)

Flag to set on the boundary prescribed final geometry

Returns

No return value

Example

To set flag f for boundary prescribed final geometry b:

```
b.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the boundary prescribed final geometry. The boundary prescribed final geometry will be sketched until you either call PrescribedFinalGeometry.Unsketch(), PrescribedFinalGeometry.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the boundary prescribed final geometry is sketched. If omitted redraw is true. If you want to sketch several boundary prescribed final geometrys and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch boundary prescribed final geometry b:

```
b.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged boundary prescribed final geometrys in the model. The boundary prescribed final geometrys will be sketched until you either call PrescribedFinalGeometry.Unsketch(), PrescribedFinalGeometry.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged boundary prescribed final geometrys will be sketched in

- **flag** (Flag)

Flag set on the boundary prescribed final geometrys that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the boundary prescribed final geometrys are sketched. If omitted redraw is true. If you want to sketch flagged boundary prescribed final geometrys several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all boundary prescribed final geometrys flagged with flag in model m:

```
PrescribedFinalGeometry.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of boundary prescribed final geometrys in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing boundary prescribed final geometrys should be counted. If false or omitted referenced but undefined boundary prescribed final geometrys will also be included in the total.

Returns

number of boundary prescribed final geometrys

Return type

Number

Example

To get the total number of boundary prescribed final geometrys in model m:

```
var total = PrescribedFinalGeometry.Total(m);
```

Unblank()

Description

Unblanks the boundary prescribed final geometry

Arguments

No arguments

Returns

No return value

Example

To unblank boundary prescribed final geometry b:

```
b.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the boundary prescribed final geometrys in the model.

Arguments

- **Model** (Model)

Model that all boundary prescribed final geometrys will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the boundary prescribed final geometrys in model m:

```
PrescribedFinalGeometry.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged boundary prescribed final geometrys in the model.

Arguments

- **Model** (Model)

Model that the flagged boundary prescribed final geometrys will be unblanked in

- **flag** (Flag)

Flag set on the boundary prescribed final geometrys that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the boundary prescribed final geometrys in model m flagged with f:

```
PrescribedFinalGeometry.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the boundary prescribed final geometrys in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all boundary prescribed final geometrys will be unset in

- **flag** (Flag)

Flag to unset on the boundary prescribed final geometrys

Returns

No return value

Example

To unset the flag f on all the boundary prescribed final geometrys in model m:

```
PrescribedFinalGeometry.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the boundary prescribed final geometry.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the boundary prescribed final geometry is unsketched. If omitted redraw is true. If you want to unsketch several boundary prescribed final geometrys and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch boundary prescribed final geometry b:

```
b.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all boundary prescribed final geometrys.

Arguments

- **Model** (Model)

Model that all boundary prescribed final geometrys will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the boundary prescribed final geometrys are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all boundary prescribed final geometrys in model m:

```
PrescribedFinalGeometry.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged boundary prescribed final geometrys in the model.

Arguments

- **Model** (Model)

Model that all boundary prescribed final geometrys will be unsketched in

- **flag** (Flag)

Flag set on the boundary prescribed final geometrys that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the boundary prescribed final geometrys are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all boundary prescribed final geometrys flagged with flag in model m:

```
PrescribedFinalGeometry.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

PrescribedFinalGeometry object.

Return type

PrescribedFinalGeometry

Example

To check if PrescribedFinalGeometry property b.example is a parameter by using the PrescribedFinalGeometry.GetParameter() method:

```
if (b.ViewParameters().GetParameter(b.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for boundary prescribed final geometry with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The boundary prescribed final geometry object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for boundary prescribed final geometry b:

```
b.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for boundary prescribed final geometry b:

```
b.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for boundary prescribed final geometry. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for boundary prescribed final geometry b:

```
b.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this boundary prescribed final geometry.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for boundary prescribed final geometry b:

```
var xrefs = b.Xrefs();
```

---

toString()

Description

Creates a string containing the PrescribedFinalGeometry data in keyword format. Note that this contains the keyword header and the keyword cards. See also PrescribedFinalGeometry.Keyword() and PrescribedFinalGeometry.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for PrescribedFinalGeometry bfg in keyword format

```
var s = bfg.toString();
```

PrescribedMotion class

The PrescribedMotion class gives you access to define boundary prescribed motion cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- AnimationBackward()
- AnimationBegin(Model*[Model]*, flag*[Flag]*)
- AnimationFinish()
- AnimationForward()
- AnimationGetData()
- AnimationPause()
- AnimationPlay()
- AnimationSetData(data*[object]*)
- AnimationToEnd()
- AnimationToStart()
- AnimationToTime()
- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()

- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

PrescribedMotion constants

Name	Description
PrescribedMotion.EDGE_UVW	Prescribed motion is *BOUNDARY_PRESCRIBED_MOTION_EDGE_UVW.
PrescribedMotion.FACE_XYZ	Prescribed motion is *BOUNDARY_PRESCRIBED_MOTION_FACE_XYZ.
PrescribedMotion.NODE	Prescribed motion is *BOUNDARY_PRESCRIBED_MOTION_NODE.
PrescribedMotion.NRBC	Prescribed motion is *BOUNDARY_PRESCRIBED_MOTION_RIGID, with an NRB, not a part.
PrescribedMotion.NRBC_LOCAL	Prescribed motion is *BOUNDARY_PRESCRIBED_MOTION_RIGID_ LOCAL, with an NRB, not a part.
PrescribedMotion.POINT_UVW	Prescribed motion is *BOUNDARY_PRESCRIBED_MOTION_POINT_UVW.
PrescribedMotion.RIGID	Prescribed motion is *BOUNDARY_PRESCRIBED_MOTION_RIGID.
PrescribedMotion.RIGID_LOCAL	Prescribed motion is *BOUNDARY_PRESCRIBED_MOTION_RIGID_ LOCAL.
PrescribedMotion.SET	Prescribed motion is *BOUNDARY_PRESCRIBED_MOTION_SET.
PrescribedMotion.SET_BOX	Prescribed motion is *BOUNDARY_PRESCRIBED_MOTION_SET_BOX.
PrescribedMotion.SET_EDGE_ UVW	Prescribed motion is *BOUNDARY_PRESCRIBED_MOTION_SET_EDGE_ UVW.
PrescribedMotion.SET_FACE_ XYZ	Prescribed motion is *BOUNDARY_PRESCRIBED_MOTION_SET_FACE_ XYZ.
PrescribedMotion.SET_LINE	Prescribed motion is *BOUNDARY_PRESCRIBED_MOTION_SET_LINE.
PrescribedMotion.SET_POINT_ UVW	Prescribed motion is *BOUNDARY_PRESCRIBED_MOTION_SET_POINT_ UVW.
PrescribedMotion.SET_ SEGMENT	Prescribed motion is *BOUNDARY_PRESCRIBED_MOTION_SET_ SEGMENT.

PrescribedMotion properties

Name	Type	Description
birth	real	Birth time
bndout2dynain	logical	true if _BNDOUT2DYNAIN option is set, false if not
death	real	Death time
dof	integer	Degree of freedom
exists (read only)	logical	true if boundary prescribed motion exists, false if referred to but not defined.
form	integer	Formulation type. Used for Card 6.
heading	string	PrescribedMotion heading

id	logical	true if _ID option is set, false if not
include	integer	The Include file number that the boundary prescribed motion is in.
label	integer	PrescribedMotion number.
lcid	integer	Load curve of motion vs. time
lrb	integer	Lead rigid body for measuring relative displacement
model (read only)	integer	The Model number that the boundary prescribed motion is in.
nbeg	integer	Node ID of a starting node. Used for PrescribedMotion.SET_LINE
nend	integer	Node ID of a ending node. Used for PrescribedMotion.SET_LINE
node1	integer	Optional orientation node for relative displacement
node2	integer	Optional orientation node for relative displacement
offset1	real	Offset 1 for types 9-11
offset2	real	Offset 2 for types 9-11
prmr	string	String representing the name of the parameter to be output to the dynain file. Used when PrescribedMotion.bndout2dynain is set to true.
sf	real	Load curve scale factor
sfd	real	Scale factor for displacement penalty stiffness. Used for Card 6.
type	constant	The Prescribed motion type. Can be PrescribedMotion.NODE, PrescribedMotion.SET, PrescribedMotion.RIGID, PrescribedMotion.RIGID_LOCAL, PrescribedMotion.NRBC, PrescribedMotion.NRBC_LOCAL, PrescribedMotion.SET_BOX, PrescribedMotion.SET_SEGMENT, PrescribedMotion.SET_LINE, PrescribedMotion.POINT_UVW, PrescribedMotion.EDGE_UVW, PrescribedMotion.FACE_XYZ, PrescribedMotion.SET_POINT_UVW, PrescribedMotion.SET_EDGE_UVW or PrescribedMotion.SET_FACE_XYZ
typeid	integer	Node ID, node set ID, part ID or NRB
vad	integer	Velocity/acceleration/displacement flag
vid	integer	Vector ID

Detailed Description

The PrescribedMotion class allows you to create, modify, edit and boundary prescribed motion cards. See the documentation below for more details.

Constructor

new PrescribedMotion(Model*[Model]*, typeid*[integer]*, dof*[integer]*, vad*[integer]*, lcid*[integer]*, type*[constant]*, label (optional)*[integer]*, heading (optional)*[string]*)

Description

Create a new PrescribedMotion object.

Arguments

- **Model** (Model)

Model that PrescribedMotion will be created in

- **typeid** (integer)

Node ID, node set ID or part ID

- **dof** (integer)

Degree of freedom

- **vad** (integer)

Velocity/acceleration/displacement flag

- **lcid** (integer)

Load curve for motion

- **type** (constant)

Specify the type of prescribed motion (Can be PrescribedMotion.NODE, PrescribedMotion.SET, PrescribedMotion.RIGID, PrescribedMotion.RIGID_LOCAL, PrescribedMotion.NRBC, PrescribedMotion.NRBC_LOCAL, PrescribedMotion.SET_BOX, PrescribedMotion.SET_SEGMENT, PrescribedMotion.SET_LINE, PrescribedMotion.POINT_UVW, PrescribedMotion.EDGE_UVW, PrescribedMotion.FACE_XYZ, PrescribedMotion.SET_POINT_UVW, PrescribedMotion.SET_EDGE_UVW or PrescribedMotion.SET_FACE_XYZ)

- **label (optional)** (integer)

PrescribedMotion number

- **heading (optional)** (string)

Title for the PrescribedMotion

Returns

PrescribedMotion object

Return type

PrescribedMotion

Example

To create a new displacement for node 100 in x using loadcurve 10 model m with label 200, of type SET

```
var b = new PrescribedMotion(m, 100, 1, 2, 10, PrescribedMotion.SET, 200);
```

Details of functions

AnimationBackward() [static]

Description

Moves backward one frame of a PrescribedMotion animation (pausing animation first if required). Also see the PrescribedMotion.AnimationBegin() method which **MUST** be called before you start animating and the PrescribedMotion.AnimationFinish() method which **MUST** be called after you have finished animating.

Arguments

No arguments

Returns

No return value

Example

To move backward one frame of an animation:

```
PrescribedMotion.AnimationBackward();
```

AnimationBegin(Model*[Model]*, flag*[Flag]*) [static]

Description

Begins a PrescribedMotion animation. This **MUST** be called before any of the other Animation methods. Also see the PrescribedMotion.AnimationFinish() method which **MUST** be called after you have finished animating.

Arguments

- **Model** (Model)

Model that PrescribedMotions are in

- **flag** (Flag)

Flag set on the PrescribedMotions that you want to animate

Returns

Object with the following properties:

Name	Type	Description
end	real	End time
frame	integer	Current frame
frames	integer	Number of frames
rate	integer	Animation speed in fps
repeat	integer	Animation repeat (0=off, 1=on)
start	real	Start time
time	real	Current time
timestep	real	Timestep

Return type

object

Example

To begin an animation of the PrescribedMotions in model m flagged with f:

```
var aprops = PrescribedMotion.AnimationBegin(m, f);
```

AnimationFinish() [static]

Description

Finishes a PrescribedMotion animation. This **MUST** be called to finish animating. This will restore nodal coordinates but will **not** perform a graphics update. Also see the PrescribedMotion.AnimationBegin() method which **MUST** be called before you start animating.

Arguments

No arguments

Returns

No return value

Example

To finish animating:

```
PrescribedMotion.AnimationFinish();
```

AnimationForward() [static]

Description

Moves forward one frame of a PrescribedMotion animation (pausing animation first if required). Also see the PrescribedMotion.AnimationBegin() method which **MUST** be called before you start animating and the PrescribedMotion.AnimationFinish() method which **MUST** be called after you have finished animating.

Arguments

No arguments

Returns

No return value

Example

To move forward one frame of an animation:

```
PrescribedMotion.AnimationForward();
```

AnimationGetData() [static]

Description

Returns the animation data (pausing animation first if required). Also see the PrescribedMotion.AnimationBegin() method which **MUST** be called before you start animating and the PrescribedMotion.AnimationFinish() method which **MUST** be called after you have finished animating.

Arguments

No arguments

Returns

Object with the following properties:

Name	Type	Description
end	real	End time
frame	integer	Current frame
frames	integer	Number of frames
rate	integer	Animation speed in fps
repeat	integer	Animation repeat (0=off, 1=on)
start	real	Start time
time	real	Current time
timestep	real	Timestep

Return type

object

Example

To get the current animation data:

```
PrescribedMotion.AnimationGetData();
```

AnimationPause() [static]

Description

Pauses playback of a PrescribedMotion animation. Also see the PrescribedMotion.AnimationBegin() method which **MUST** be called before you start animating and the PrescribedMotion.AnimationFinish() method which **MUST** be called after you have finished animating.

Arguments

No arguments

Returns

No return value

Example

To pause playback of an animation:

```
PrescribedMotion.AnimationPause();
```

AnimationPlay() [static]

Description

Starts playback of a PrescribedMotion animation. Also see the PrescribedMotion.AnimationBegin() method which **MUST** be called before you start animating and the PrescribedMotion.AnimationFinish() method which **MUST** be called after you have finished animating.
This method should only be used from a script which implements a user interface so you can actually stop the animation! Don't forget to add a pause/stop button that calls PrescribedMotion.AnimationPause()!

Arguments

No arguments

Returns

No return value

Example

To start playback of an animation:

```
PrescribedMotion.AnimationPlay();
```

AnimationSetData(data*[object]*) [static]

Description

Sets the current animation data (pausing animation first if required). Also see the PrescribedMotion.AnimationBegin() method which **MUST** be called before you start animating and the PrescribedMotion.AnimationFinish() method which **MUST** be called after you have finished animating.

Arguments

- **data** (object)

data returned from PrescribedMotion.AnimationBegin() or PrescribedMotion.AnimationGetData()

Object has the following properties:

Name	Type	Description
end	real	End time
frame	integer	Current frame

frames	integer	Number of frames
rate	integer	Animation speed in fps
repeat	integer	Animation repeat (0=off, 1=on)
start	real	Start time
time	real	Current time
timestep	real	Timestep

Returns

No return value

Example

To set the animation frame rate to 10 frames/sec:

```
data = PrescribedMotion.AnimationGetData();
data.rate = 10;
PrescribedMotion.AnimationSetData(data);
```

AnimationToEnd() [static]

Description

Moves to the end of a PrescribedMotion animation (pausing animation first if required). Also see the PrescribedMotion.AnimationBegin() method which **MUST** be called before you start animating and the PrescribedMotion.AnimationFinish() method which **MUST** be called after you have finished animating.

Arguments

No arguments

Returns

No return value

Example

To move to the end of an animation:

```
PrescribedMotion.AnimationToEnd();
```

AnimationToStart() [static]

Description

Moves to the start of a PrescribedMotion animation (pausing animation first if required). Also see the PrescribedMotion.AnimationBegin() method which **MUST** be called before you start animating and the PrescribedMotion.AnimationFinish() method which **MUST** be called after you have finished animating.

Arguments

No arguments

Returns

No return value

Example

To move to the start of an animation:

```
PrescribedMotion.AnimationToStart();
```

AnimationToTime() [static]

Description

Moves to a specific time in a PrescribedMotion animation (pausing animation first if required). Also see the PrescribedMotion.AnimationBegin() method which **MUST** be called before you start animating and the PrescribedMotion.AnimationFinish() method which **MUST** be called after you have finished animating.

Arguments

No arguments

Returns

No return value

Example

To move to time 28.0 in an animation:

```
PrescribedMotion.AnimationToTime(28.0);
```

AssociateComment(Comment[*Comment*])

Description

Associates a comment with a boundary prescribed motion.

Arguments

- **Comment** (Comment)

Comment that will be attached to the boundary prescribed motion

Returns

No return value

Example

To associate comment c to the boundary prescribed motion b:

```
b.AssociateComment(c);
```

Blank()

Description

Blanks the boundary prescribed motion

Arguments

No arguments

Returns

No return value

Example

To blank boundary prescribed motion b:

```
b.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the boundary prescribed motions in the model.

Arguments

- **Model** (Model)

Model that all boundary prescribed motions will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the boundary prescribed motions in model m:

```
PrescribedMotion.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged boundary prescribed motions in the model.

Arguments

- **Model** (Model)

Model that all the flagged boundary prescribed motions will be blanked in

- **flag** (Flag)

Flag set on the boundary prescribed motions that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the boundary prescribed motions in model m flagged with f:

```
PrescribedMotion.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the boundary prescribed motion is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if boundary prescribed motion b is blanked:

```
if (b.Blanked() ) do_something...
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the boundary prescribed motion.

Arguments

- **flag** (Flag)

Flag to clear on the boundary prescribed motion

Returns

No return value

Example

To clear flag f for boundary prescribed motion b:

```
b.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the boundary prescribed motion. The target include of the copied boundary prescribed motion can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

PrescribedMotion object

Return type

PrescribedMotion

Example

To copy boundary prescribed motion b into boundary prescribed motion z:

```
var z = b.Copy();
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a boundary prescribed motion.

Arguments

- **Comment** (Comment)

Comment that will be detached from the boundary prescribed motion

Returns

No return value

Example

To detach comment c from the boundary prescribed motion b:

```
b.DetachComment(c);
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for boundary prescribed motion with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The boundary prescribed motion object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for boundary prescribed motion b:

```
b.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for boundary prescribed motion b:

```
b.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for boundary prescribed motion. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for boundary prescribed motion b:

```
b.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first boundary prescribed motion in the model.

Arguments

- **Model** (Model)

Model to get first boundary prescribed motion in

Returns

PrescribedMotion object (or null if there are no boundary prescribed motions in the model).

Return type

PrescribedMotion

Example

To get the first boundary prescribed motion in model m:

```
var b = PrescribedMotion.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free boundary prescribed motion label in the model. Also see PrescribedMotion.LastFreeLabel(), PrescribedMotion.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free boundary prescribed motion label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

PrescribedMotion label.

Return type

Number

Example

To get the first free boundary prescribed motion label in model m:

```
var label = PrescribedMotion.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the boundary prescribed motions in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all boundary prescribed motions will be flagged in

- **flag** (Flag)

Flag to set on the boundary prescribed motions

Returns

No return value

Example

To flag all of the boundary prescribed motions with flag f in model m:

```
PrescribedMotion.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the boundary prescribed motion is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the boundary prescribed motion

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if boundary prescribed motion b has flag f set on it:

```
if (b.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each boundary prescribed motion in the model.
Note that ForEach has been designed to make looping over boundary prescribed motions as fast as possible and so has some limitations.
Firstly, a single temporary PrescribedMotion object is created and on each function call it is updated with the current boundary prescribed motion data. This means that you should not try to store the PrescribedMotion object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new boundary prescribed motions inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all boundary prescribed motions are in

- **func** (function)

Function to call for each boundary prescribed motion

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the boundary prescribed motions in model m:

```
PrescribedMotion.ForEach(m, test);
function test(b)
{
// b is PrescribedMotion object
}
```

To call function test for all of the boundary prescribed motions in model m with optional object:

```
var data = { x:0, y:0 };
PrescribedMotion.ForEach(m, test, data);
function test(b, extra)
{
// b is PrescribedMotion object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of PrescribedMotion objects or properties for all of the boundary prescribed motions in a model in PRIMER. If the optional property argument is not given then an array of PrescribedMotion objects is returned. If the property argument is given, that property value for each boundary prescribed motion is returned in the array instead of a PrescribedMotion object

Arguments

- **Model** (Model)

Model to get boundary prescribed motions from

- **property (optional)** (string)

Name for property to get for all boundary prescribed motions in the model

Returns

Array of PrescribedMotion objects or properties

Return type

Array

Example

To make an array of PrescribedMotion objects for all of the boundary prescribed motions in model m:

```
var a = PrescribedMotion.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each boundary prescribed motion in model m:

```
var a = PrescribedMotion.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a boundary prescribed motion.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the boundary prescribed motion b:

```
var comm_array = b.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of PrescribedMotion objects for all of the flagged boundary prescribed motions in a model in PRIMER If the optional property argument is not given then an array of PrescribedMotion objects is returned. If the property argument is given, then that property value for each boundary prescribed motion is returned in the array instead of a PrescribedMotion object

Arguments

- **Model** (Model)

Model to get boundary prescribed motions from

- **flag** (Flag)

Flag set on the boundary prescribed motions that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged boundary prescribed motions in the model

Returns

Array of PrescribedMotion objects or properties

Return type

Array

Example

To make an array of PrescribedMotion objects for all of the boundary prescribed motions in model m flagged with f:

```
var b = PrescribedMotion.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the boundary prescribed motions in model m flagged with f:

```
var a = PrescribedMotion.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the PrescribedMotion object for a boundary prescribed motion ID.

Arguments

- **Model** (Model)

Model to find the boundary prescribed motion in

- **number** (integer)

number of the boundary prescribed motion you want the PrescribedMotion object for

Returns

PrescribedMotion object (or null if boundary prescribed motion does not exist).

Return type

PrescribedMotion

Example

To get the PrescribedMotion object for boundary prescribed motion 100 in model m

```
var b = PrescribedMotion.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a PrescribedMotion property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the PrescribedMotion.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

boundary prescribed motion property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if PrescribedMotion property b.example is a parameter:

```
Options.property_parameter_names = true;
if (b.GetParameter(b.example) ) do_something...
Options.property_parameter_names = false;
```

To check if PrescribedMotion property b.example is a parameter by using the GetParameter method:

```
if (b.ViewParameters().GetParameter(b.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this PrescribedMotion (*BOUNDARY_PRESCRIBED_MOTION_xxxx). **Note that a carriage return is not added**. See also PrescribedMotion.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for PrescribedMotion pm:

```
var key = pm.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the PrescribedMotion. **Note that a carriage return is not added**. See also PrescribedMotion.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for PrescribedMotion pm:

```
var cards = pm.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last boundary prescribed motion in the model.

Arguments

- **Model** (Model)

Model to get last boundary prescribed motion in

Returns

PrescribedMotion object (or null if there are no boundary prescribed motions in the model).

Return type

PrescribedMotion

Example

To get the last boundary prescribed motion in model m:

```
var b = PrescribedMotion.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free boundary prescribed motion label in the model. Also see PrescribedMotion.FirstFreeLabel(), PrescribedMotion.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free boundary prescribed motion label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

PrescribedMotion label.

Return type

Number

Example

To get the last free boundary prescribed motion label in model m:

```
var label = PrescribedMotion.LastFreeLabel(m);
```

Next()

Description

Returns the next boundary prescribed motion in the model.

Arguments

No arguments

Returns

PrescribedMotion object (or null if there are no more boundary prescribed motions in the model).

Return type

PrescribedMotion

Example

To get the boundary prescribed motion in model m after boundary prescribed motion b:

```
var b = b.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) boundary prescribed motion label in the model. Also see PrescribedMotion.FirstFreeLabel(), PrescribedMotion.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free boundary prescribed motion label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

PrescribedMotion label.

Return type

Number

Example

To get the next free boundary prescribed motion label in model m:

```
var label = PrescribedMotion.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a boundary prescribed motion.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only boundary prescribed motions from that model can be picked. If the argument is a Flag then only boundary prescribed motions that are flagged with *limit* can be selected. If omitted, or null, any boundary prescribed motions from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

PrescribedMotion object (or null if not picked)

Return type

PrescribedMotion

Example

To pick a boundary prescribed motion from model m giving the prompt 'Pick boundary prescribed motion from screen':

```
var b = PrescribedMotion.Pick('Pick boundary prescribed motion from screen', m);
```

Previous()

Description

Returns the previous boundary prescribed motion in the model.

Arguments

No arguments

Returns

PrescribedMotion object (or null if there are no more boundary prescribed motions in the model).

Return type

PrescribedMotion

Example

To get the boundary prescribed motion in model m before boundary prescribed motion b:

```
var b = b.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the boundary prescribed motions in the model.

Arguments

- **Model** (Model)

Model that all boundary prescribed motions will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the boundary prescribed motions in model m, from 1000000:

```
PrescribedMotion.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged boundary prescribed motions in the model.

Arguments

- **Model** (Model)

Model that all the flagged boundary prescribed motions will be renumbered in

- **flag** (Flag)

Flag set on the boundary prescribed motions that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the boundary prescribed motions in model m flagged with f, from 1000000:

```
PrescribedMotion.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select boundary prescribed motions using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting boundary prescribed motions

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only boundary prescribed motions from that model can be selected. If the argument is a Flag then only boundary prescribed motions that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any boundary prescribed motions can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of boundary prescribed motions selected or null if menu cancelled

Return type

Number

Example

To select boundary prescribed motions from model m, flagging those selected with flag f, giving the prompt 'Select boundary prescribed motions':

```
PrescribedMotion.Select(f, 'Select boundary prescribed motions', m);
```

To select boundary prescribed motions, flagging those selected with flag f but limiting selection to boundary prescribed motions flagged with flag l, giving the prompt 'Select boundary prescribed motions':

```
PrescribedMotion.Select(f, 'Select boundary prescribed motions', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the boundary prescribed motion.

Arguments

- **flag** (Flag)

Flag to set on the boundary prescribed motion

Returns

No return value

Example

To set flag f for boundary prescribed motion b:

```
b.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the boundary prescribed motion. The boundary prescribed motion will be sketched until you either call PrescribedMotion.Unsketch(), PrescribedMotion.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the boundary prescribed motion is sketched. If omitted redraw is true. If you want to sketch several boundary prescribed motions and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch boundary prescribed motion b:

```
b.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged boundary prescribed motions in the model. The boundary prescribed motions will be sketched until you either call PrescribedMotion.Unsketch(), PrescribedMotion.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged boundary prescribed motions will be sketched in

- **flag** (Flag)

Flag set on the boundary prescribed motions that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the boundary prescribed motions are sketched. If omitted redraw is true. If you want to sketch flagged boundary prescribed motions several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all boundary prescribed motions flagged with flag in model m:

```
PrescribedMotion.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of boundary prescribed motions in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing boundary prescribed motions should be counted. If false or omitted referenced but undefined boundary prescribed motions will also be included in the total.

Returns

number of boundary prescribed motions

Return type

Number

Example

To get the total number of boundary prescribed motions in model m:

```
var total = PrescribedMotion.Total(m);
```

Unblank()

Description

Unblanks the boundary prescribed motion

Arguments

No arguments

Returns

No return value

Example

To unblank boundary prescribed motion b:

```
b.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the boundary prescribed motions in the model.

Arguments

- **Model** (Model)

Model that all boundary prescribed motions will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the boundary prescribed motions in model m:

```
PrescribedMotion.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged boundary prescribed motions in the model.

Arguments

- **Model** (Model)

Model that the flagged boundary prescribed motions will be unblanked in

- **flag** (Flag)

Flag set on the boundary prescribed motions that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the boundary prescribed motions in model m flagged with f:

```
PrescribedMotion.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the boundary prescribed motions in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all boundary prescribed motions will be unset in

- **flag** (Flag)

Flag to unset on the boundary prescribed motions

Returns

No return value

Example

To unset the flag f on all the boundary prescribed motions in model m:

```
PrescribedMotion.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the boundary prescribed motion.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the boundary prescribed motion is unsketched. If omitted redraw is true. If you want to unsketch several boundary prescribed motions and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch boundary prescribed motion b:

```
b.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all boundary prescribed motions.

Arguments

- **Model** (Model)

Model that all boundary prescribed motions will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the boundary prescribed motions are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all boundary prescribed motions in model m:

```
PrescribedMotion.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged boundary prescribed motions in the model.

Arguments

- **Model** (Model)

Model that all boundary prescribed motions will be unsketched in

- **flag** (Flag)

Flag set on the boundary prescribed motions that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the boundary prescribed motions are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all boundary prescribed motions flagged with flag in model m:

```
PrescribedMotion.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

PrescribedMotion object.

Return type

PrescribedMotion

Example

To check if PrescribedMotion property b.example is a parameter by using the PrescribedMotion.GetParameter() method:

```
if (b.ViewParameters().GetParameter(b.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for boundary prescribed motion with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The boundary prescribed motion object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for boundary prescribed motion b:

```
b.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for boundary prescribed motion b:

```
b.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for boundary prescribed motion. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for boundary prescribed motion b:

```
b.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this boundary prescribed motion.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for boundary prescribed motion b:

```
var xrefs = b.Xrefs();
```

---

toString()

Description

Creates a string containing the PrescribedMotion data in keyword format. Note that this contains the keyword header and the keyword cards. See also PrescribedMotion.Keyword() and PrescribedMotion.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for PrescribedMotion pm in keyword format

```
var s = pm.toString();
```

Spc class

The Spc class gives you access to define spc cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(type*[integer]*, Model*[Model]*, flag*[Flag]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(type*[constant]*, redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

Spc constants

Constants for sketching mode

Name	Description
Spc.ROTATIONAL	Sketch rotational degrees of freedom.
Spc.TRANSLATIONAL	Sketch translational degrees of freedom.

Constants for suffix

Name	Description
Spc.NODE	SPC is *BOUNDARY_SPC_NODE.
Spc.SET	SPC is *BOUNDARY_SPC_SET.

Spc properties

Name	Type	Description
cid	integer	Coordinate system ID
dofrx	integer	Rotational constraint in local x direction
dofry	integer	Rotational constraint in local y direction
dofrz	integer	Rotational constraint in local z direction
dofx	integer	Translational constraint in local x direction
dofy	integer	Translational constraint in local y direction
dofz	integer	Translational constraint in local z direction
exists (read only)	logical	true if boundary SPC exists, false if referred to but not defined.
heading	string	Spc heading
id	logical	true if _ID option is set, false if not.
include	integer	The Include file number that the boundary SPC is in.
label	integer	Spc number.
model (read only)	integer	The Model number that the boundary SPC is in.
nid	integer	Node ID or node set ID
type	constant	The Spc type. Can be Spc.NODE or Spc.SET.

Properties for _BIRTH_DEATH option

Name	Type	Description
bd_flag	logical	true if _BIRTH_DEATH option is set, false if not
birth	real	Activation time for constraint
death	real	Deactivation time for constraint

Detailed Description

The Spc class allows you to create, modify, edit and manipulate spc cards. See the documentation below for more details.

Constructor

new Spc(Model*[Model]*, nid*[integer]*, cid*[integer]*, dofx*[integer]*, dofy*[integer]*, dofz*[integer]*, dofrx*[integer]*, dofry*[integer]*, dofrz*[integer]*, type*[constant]*, label (optional)*[integer]*, heading (optional)*[string]*)

Description

Create a new Spc object.

Arguments

- **Model** (Model)

Model that spc will be created in

- **nid** (integer)

Node ID or node set ID

- **cid** (integer)

Coordinate system ID

- **dofx** (integer)

Translational constraint in local x direction

- **dofy** (integer)

Translational constraint in local y direction

- **dofz** (integer)

Translational constraint in local z direction

- **dofrx** (integer)

Rotational constraint in local x direction

- **dofry** (integer)

Rotational constraint in local y direction

- **dofrz** (integer)

Rotational constraint in local z direction

- **type** (constant)

Specify the type of boundary spc (Can be Spc.NODE or Spc.SET)

- **label (optional)** (integer)

Spc number

- **heading (optional)** (string)

Title for the spc

Returns

Spc object

Return type

Spc

Example

To create a new boundary spc in model m with label 200, of type SET

```
var b = new Spc(m, 200, 0, 1, 0, 0, 1, 0, 0, Spc.SET, 200);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a boundary SPC.

Arguments

- **Comment** (Comment)

Comment that will be attached to the boundary SPC

Returns

No return value

Example

To associate comment c to the boundary SPC s:

```
s.AssociateComment(c);
```

Blank()

Description

Blanks the boundary SPC

Arguments

No arguments

Returns

No return value

Example

To blank boundary SPC s:

```
s.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the boundary SPCs in the model.

Arguments

- **Model** (Model)

Model that all boundary SPCs will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the boundary SPCs in model m:

```
Spc.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged boundary SPCs in the model.

Arguments

- **Model** (Model)

Model that all the flagged boundary SPCs will be blanked in

- **flag** (Flag)

Flag set on the boundary SPCs that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the boundary SPCs in model m flagged with f:

```
Spc.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the boundary SPC is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if boundary SPC s is blanked:

```
if (s.Blanked() ) do_something...
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the boundary SPC.

Arguments

- **flag** (Flag)

Flag to clear on the boundary SPC

Returns

No return value

Example

To clear flag f for boundary SPC s:

```
s.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the boundary SPC. The target include of the copied boundary SPC can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

Spc object

Return type

Spc

Example

To copy boundary SPC s into boundary SPC z:

```
var z = s.Copy();
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a boundary SPC.

Arguments

- **Comment** (Comment)

Comment that will be detached from the boundary SPC

Returns

No return value

Example

To detach comment c from the boundary SPC s:

```
s.DetachComment(c);
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for boundary SPC with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The boundary SPC object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for boundary SPC s:

```
s.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for boundary SPC s:

```
s.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for boundary SPC. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for boundary SPC s:

```
s.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first boundary SPC in the model.

Arguments

- **Model** (Model)

Model to get first boundary SPC in

Returns

Spc object (or null if there are no boundary SPCs in the model).

Return type

Spc

Example

To get the first boundary SPC in model m:

```
var s = Spc.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free boundary SPC label in the model. Also see Spc.LastFreeLabel(), Spc.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free boundary SPC label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

Spc label.

Return type

Number

Example

To get the first free boundary SPC label in model m:

```
var label = Spc.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the boundary SPCs in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all boundary SPCs will be flagged in

- **flag** (Flag)

Flag to set on the boundary SPCs

Returns

No return value

Example

To flag all of the boundary SPCs with flag f in model m:

```
Spc.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the boundary SPC is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the boundary SPC

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if boundary SPC s has flag f set on it:

```
if (s.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each boundary SPC in the model.
Note that ForEach has been designed to make looping over boundary SPCs as fast as possible and so has some limitations.
Firstly, a single temporary Spc object is created and on each function call it is updated with the current boundary SPC data. This means that you should not try to store the Spc object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new boundary SPCs inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all boundary SPCs are in

- **func** (function)

Function to call for each boundary SPC

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the boundary SPCs in model m:

```
Spc.ForEach(m, test);
function test(s)
{
// s is Spc object
}
```

To call function test for all of the boundary SPCs in model m with optional object:

```
var data = { x:0, y:0 };
Spc.ForEach(m, test, data);
function test(s, extra)
{
// s is Spc object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Spc objects or properties for all of the boundary SPCs in a model in PRIMER. If the optional property argument is not given then an array of Spc objects is returned. If the property argument is given, that property value for each boundary SPC is returned in the array instead of a Spc object

Arguments

- **Model** (Model)

Model to get boundary SPCs from

- **property (optional)** (string)

Name for property to get for all boundary SPCs in the model

Returns

Array of Spc objects or properties

Return type

Array

Example

To make an array of Spc objects for all of the boundary SPCs in model m:

```
var a = Spc.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each boundary SPC in model m:

```
var a = Spc.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a boundary SPC.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the boundary SPC s:

```
var comm_array = s.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of Spc objects for all of the flagged boundary SPCs in a model in PRIMER If the optional property argument is not given then an array of Spc objects is returned. If the property argument is given, then that property value for each boundary SPC is returned in the array instead of a Spc object

Arguments

- **Model** (Model)

Model to get boundary SPCs from

- **flag** (Flag)

Flag set on the boundary SPCs that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged boundary SPCs in the model

Returns

Array of Spc objects or properties

Return type

Array

Example

To make an array of Spc objects for all of the boundary SPCs in model m flagged with f:

```
var s = Spc.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the boundary SPCs in model m flagged with f:

```
var a = Spc.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the Spc object for a boundary SPC ID.

Arguments

- **Model** (Model)

Model to find the boundary SPC in

- **number** (integer)

number of the boundary SPC you want the Spc object for

Returns

Spc object (or null if boundary SPC does not exist).

Return type

Spc

Example

To get the Spc object for boundary SPC 100 in model m

```
var s = Spc.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a Spc property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the Spc.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

boundary SPC property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if Spc property s.example is a parameter:

```
Options.property_parameter_names = true;
if (s.GetParameter(s.example) ) do_something...
Options.property_parameter_names = false;
```

To check if Spc property s.example is a parameter by using the GetParameter method:

```
if (s.ViewParameters().GetParameter(s.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this spc (*BOUNDARY_SPC_xxxx). **Note that a carriage return is not added**. See also Spc.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for spc s:

```
var key = s.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the spc. **Note that a carriage return is not added**. See also Spc.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for spc s:

```
var cards = s.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last boundary SPC in the model.

Arguments

- **Model** (Model)

Model to get last boundary SPC in

Returns

Spc object (or null if there are no boundary SPCs in the model).

Return type

Spc

Example

To get the last boundary SPC in model m:

```
var s = Spc.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free boundary SPC label in the model. Also see Spc.FirstFreeLabel(), Spc.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free boundary SPC label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

Spc label.

Return type

Number

Example

To get the last free boundary SPC label in model m:

```
var label = Spc.LastFreeLabel(m);
```

Next()

Description

Returns the next boundary SPC in the model.

Arguments

No arguments

Returns

Spc object (or null if there are no more boundary SPCs in the model).

Return type

Spc

Example

To get the boundary SPC in model m after boundary SPC s:

```
var s = s.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) boundary SPC label in the model. Also see Spc.FirstFreeLabel(), Spc.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free boundary SPC label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

Spc label.

Return type

Number

Example

To get the next free boundary SPC label in model m:

```
var label = Spc.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a boundary SPC.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only boundary SPCs from that model can be picked. If the argument is a Flag then only boundary SPCs that are flagged with *limit* can be selected. If omitted, or null, any boundary SPCs from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

Spc object (or null if not picked)

Return type

Spc

Example

To pick a boundary SPC from model m giving the prompt 'Pick boundary SPC from screen':

```
var s = Spc.Pick('Pick boundary SPC from screen', m);
```

Previous()

Description

Returns the previous boundary SPC in the model.

Arguments

No arguments

Returns

Spc object (or null if there are no more boundary SPCs in the model).

Return type

Spc

Example

To get the boundary SPC in model m before boundary SPC s:

```
var s = s.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the boundary SPCs in the model.

Arguments

- **Model** (Model)

Model that all boundary SPCs will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the boundary SPCs in model m, from 1000000:

```
Spc.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged boundary SPCs in the model.

Arguments

- **Model** (Model)

Model that all the flagged boundary SPCs will be renumbered in

- **flag** (Flag)

Flag set on the boundary SPCs that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the boundary SPCs in model m flagged with f, from 1000000:

```
Spc.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select boundary SPCs using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting boundary SPCs

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only boundary SPCs from that model can be selected. If the argument is a Flag then only boundary SPCs that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any boundary SPCs can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of boundary SPCs selected or null if menu cancelled

Return type

Number

Example

To select boundary SPCs from model m, flagging those selected with flag f, giving the prompt 'Select boundary SPCs':

```
Spc.Select(f, 'Select boundary SPCs', m);
```

To select boundary SPCs, flagging those selected with flag f but limiting selection to boundary SPCs flagged with flag l, giving the prompt 'Select boundary SPCs':

```
Spc.Select(f, 'Select boundary SPCs', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the boundary SPC.

Arguments

- **flag** (Flag)

Flag to set on the boundary SPC

Returns

No return value

Example

To set flag f for boundary SPC s:

```
s.SetFlag(f);
```

Sketch(type*[constant]*, redraw (optional)*[boolean]*)

Description

Sketches the Boundary SPC. The SPC will be sketched until you do a graphics update or delete the model

Arguments

- **type** (constant)

Type of constraints to be drawn. Can be Spc.TRANSLATIONAL or Spc.ROTATIONAL.

- **redraw (optional)** (boolean)

If set to true (or omitted) the plot will be redrawn each time. If sketching a large number of items, efficiency will be gained by setting the argument to false for all but the last item sketched. The final call will redraw.

Returns

No return value

Example

To sketch SPC s - Translational constraint

```
s1.Sketch(Spc.TRANSLATIONAL, false);
s2.Sketch(Spc.TRANSLATIONAL, false);
s3.Sketch(Spc.TRANSLATIONAL, true);
```

SketchFlagged(type*[integer]*, Model*[Model]*, flag*[Flag]*) [static]

Description

Sketches all the flagged boundary SPCs in the model and update the plot. The SPCs will be sketched until you do a graphics update or delete the model.

Arguments

- **type** (integer)

Type of constraints to be drawn. Can be Spc.TRANSLATIONAL or Spc.ROTATIONAL.

- **Model** (Model)

Model that all the flagged boundary SPCs will be sketched in

- **flag** (Flag)

Flag set on the boundary SPCs that you want to sketch

Returns

No return value

Example

To sketch translational SPCs flagged with f in model m and redraw

```
Spc.SketchFlagged(Spc.TRANSLATIONAL, m, f);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of boundary SPCs in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing boundary SPCs should be counted. If false or omitted referenced but undefined boundary SPCs will also be included in the total.

Returns

number of boundary SPCs

Return type

Number

Example

To get the total number of boundary SPCs in model m:

```
var total = Spc.Total(m);
```

Unblank()

Description

Unblanks the boundary SPC

Arguments

No arguments

Returns

No return value

Example

To unblank boundary SPC s:

```
s.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the boundary SPCs in the model.

Arguments

- **Model** (Model)

Model that all boundary SPCs will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the boundary SPCs in model m:

```
Spc.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged boundary SPCs in the model.

Arguments

- **Model** (Model)

Model that the flagged boundary SPCs will be unblanked in

- **flag** (Flag)

Flag set on the boundary SPCs that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the boundary SPCs in model m flagged with f:

```
Spc.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the boundary SPCs in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all boundary SPCs will be unset in

- **flag** (Flag)

Flag to unset on the boundary SPCs

Returns

No return value

Example

To unset the flag f on all the boundary SPCs in model m:

```
Spc.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the Spc.

Arguments

- **redraw (optional)** (boolean)

If set to true (or omitted) the plot will be redrawn each time. If unsketching a large number of items, efficiency will be gained by setting the argument to false for all but the last item unsketched. The final call will redraw.

Returns

No return value

Example

To unsketch SPC s:

```
s.Unsketch();
```

UnsketchAll(Model*[Model]*) [static]

Description

Unsketches all SPCs.

Arguments

- **Model** (Model)

Model that all SPCs will be unblanked in

Returns

No return value

Example

To unsketch all SPCs in model m and redraw:

```
SPC.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsketches all flagged SPCs.

Arguments

- **Model** (Model)

Model that all SPCs will be unsketched in

- **flag** (Flag)

Flag set on the SPCs that you want to unsketch

Returns

No return value

Example

To unsketch all SPCs in model m which are flagged with f and redraw:

```
SPC.UnsketchFlagged(m, f);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

Spc object.

Return type

Spc

Example

To check if Spc property s.example is a parameter by using the Spc.GetParameter() method:

```
if (s.ViewParameters().GetParameter(s.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for boundary SPC with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The boundary SPC object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for boundary SPC s:

```
s.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for boundary SPC s:

```
s.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for boundary SPC. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for boundary SPC s:

```
s.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this boundary SPC.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for boundary SPC s:

```
var xrefs = s.Xrefs();
```

---

toString()

Description

Creates a string containing the spc data in keyword format. Note that this contains the keyword header and the keyword cards. See also Spc.Keyword() and Spc.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for spc s in keyword format

```
var str = s.toString();
```

Comment class

The Comment class gives you access to comment cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)

Member functions

- AddLine(Line content*[String or array of strings]*, Line number (optional)*[Integer]*)
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DeleteLine(Line number*[Integer]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetLine(Line (optional)*[integer]*)
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- ModifyLine(Line number*[Integer]*, New line content*[String]*)
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

Comment constants

Constants for Comment anchor_mode types

Name	Description
Comment.MULTIPLE	The *COMMENT is associated with all cards in the next block of keywords.
Comment.SINGLE	The *COMMENT is associated with just the one immediately following keyword.

Comment properties

Name	Type	Description
anchor_mode	integer	Anchor mode. Can be Comment.SINGLE, Comment.MULTIPLE.
exists (read only)	logical	true if comment exists, false if referred to but not defined.
header	string	The header of the comment, or empty if the comment has no header.
include	integer	The Include file number that the comment is in.
model (read only)	integer	The Model number that the comment is in.
nlines	integer	Number of lines in the comment.
noecho	logical	true if _NOECHO option is set, false if not.

Detailed Description

The Comment class allows you to create, modify, edit and manipulate comment cards. See the documentation below for more details.

Constructor

new Comment(Model*[Model]*, Header (optional)*[string]*, Mode (optional)*[constant]*)

Description

Create a new Comment object.

Arguments

- **Model** (Model)

Model that comment will be created in

- **Header (optional)** (string)

Comment number

- **Mode (optional)** (constant)

Anchor: single or multiple

Returns

Comment object

Return type

Comment

Example

To create a new comment in model m with header "My header", and multiple anchor:

```
var c = new Comment(m, "My header", Comment.MULTIPLE);
```

To create a new comment in model m without header, and single anchor:

```
var c = new Comment(m);
```

Details of functions

AddLine(Line content*[String or array of strings]*, Line number (optional)*[Integer]*)

Description

Adds a line, or an array of lines, to a comment object.

Arguments

- **Line content** (String or array of strings)

String that will be added to a line

- **Line number (optional)** (Integer)

0: First line, 1: Second line, etc.

If array of lines has been passed in the first argument, the first line of the array will be inserted in the line number specified in second argument, the second line of the array will be inserted in the following line number, etc.

If that line already exists, that line and rest of them below will be shifted down.

If greater than number of existing lines, blank lines will be added.

If lower than 0, not valid argument.

If no argument, the line(s) will be appended at the end.

Returns

no return value

Example

To add a new line in the second row of comment c:

```
var str = c.AddLine("New line", 1);
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse comment c:

```
c.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the comment.

Arguments

- **flag** (Flag)

Flag to clear on the comment

Returns

No return value

Example

To clear flag f for comment c:

```
c.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the comment. The target include of the copied comment can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

Comment object

Return type

Comment

Example

To copy comment c into comment z:

```
var z = c.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a comment

Arguments

- **Model** (Model)

Model that the comment will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

Comment object (or null if not made).

Return type

Comment

Example

To start creating a comment in model m:

```
var c = Comment.Create(m);
```

DeleteLine(Line number*[Integer]*)

Description

Deletes a line of a comment.

Arguments

- **Line number** (Integer)

Line number to delete (starting at 0). The following lines will be shifted up.

Returns

no return value

Example

To delete the line in the second row of comment c:

```
var str = c.DeleteLine(1);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit comment c:

```
c.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for comment with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The comment object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for comment c:

```
c.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for comment c:

```
c.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for comment. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for comment c:

```
c.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first comment in the model.

Arguments

- **Model** (Model)

Model to get first comment in

Returns

Comment object (or null if there are no comments in the model).

Return type

Comment

Example

To get the first comment in model m:

```
var c = Comment.First(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the comments in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all comments will be flagged in

- **flag** (Flag)

Flag to set on the comments

Returns

No return value

Example

To flag all of the comments with flag f in model m:

```
Comment.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the comment is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the comment

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if comment c has flag f set on it:

```
if (c.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each comment in the model.
Note that ForEach has been designed to make looping over comments as fast as possible and so has some limitations.
Firstly, a single temporary Comment object is created and on each function call it is updated with the current comment data. This means that you should not try to store the Comment object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new comments inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all comments are in

- **func** (function)

Function to call for each comment

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the comments in model m:

```
Comment.ForEach(m, test);
function test(c)
{
// c is Comment object
}
```

To call function test for all of the comments in model m with optional object:

```
var data = { x:0, y:0 };
Comment.ForEach(m, test, data);
function test(c, extra)
{
// c is Comment object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Comment objects or properties for all of the comments in a model in PRIMER. If the optional property argument is not given then an array of Comment objects is returned. If the property argument is given, that property value for each comment is returned in the array instead of a Comment object

Arguments

- **Model** (Model)

Model to get comments from

- **property (optional)** (string)

Name for property to get for all comments in the model

Returns

Array of Comment objects or properties

Return type

Array

Example

To make an array of Comment objects for all of the comments in model m:

```
var a = Comment.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each comment in model m:

```
var a = Comment.GetAll(m, 'foo');
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of Comment objects for all of the flagged comments in a model in PRIMER If the optional property argument is not given then an array of Comment objects is returned. If the property argument is given, then that property value for each comment is returned in the array instead of a Comment object

Arguments

- **Model** (Model)

Model to get comments from

- **flag** (Flag)

Flag set on the comments that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged comments in the model

Returns

Array of Comment objects or properties

Return type

Array

Example

To make an array of Comment objects for all of the comments in model m flagged with f:

```
var c = Comment.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the comments in model m flagged with f:

```
var a = Comment.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the Comment object for a comment ID.

Arguments

- **Model** (Model)

Model to find the comment in

- **number** (integer)

number of the comment you want the Comment object for

Returns

Comment object (or null if comment does not exist).

Return type

Comment

Example

To get the Comment object for comment 100 in model m

```
var c = Comment.GetFromID(m, 100);
```

GetLine(Line (optional)*[integer]*)

Description

Extracts the lines (the strings) from a comment object.

Arguments

- **Line (optional)** (integer)

Line number to be extracted. Default value: 0 (first line)

Returns

String (or null if no lines in the comment and not argument passed)

Return type

String

Example

To extract the first line of comment c:

```
var str = c.GetLine();
```

GetParameter(prop*[string]*)

Description

Checks if a Comment property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the Comment.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

comment property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if Comment property c.example is a parameter:

```
Options.property_parameter_names = true;
if (c.GetParameter(c.example) ) do_something...
Options.property_parameter_names = false;
```

To check if Comment property c.example is a parameter by using the GetParameter method:

```
if (c.ViewParameters().GetParameter(c.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this comment (*COMMENT) and the header of the comment if there is one. **Note that a carriage return is not added**. See also Comment.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for comment c:

```
var key = c.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the comment. **Note that a carriage return is not added**. See also Comment.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for comment c:

```
var cards = c.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last comment in the model.

Arguments

- **Model** (Model)

Model to get last comment in

Returns

Comment object (or null if there are no comments in the model).

Return type

Comment

Example

To get the last comment in model m:

```
var c = Comment.Last(m);
```

ModifyLine(Line number*[Integer]*, New line content*[String]*)

Description

Modifies the content of a line in a comment.

Arguments

- **Line number** (Integer)

Line number to modify (starting at 0)

- **New line content** (String)

String that replaces the existing one in a line

Returns

no return value

Example

To modify the line in the second row of comment c:

```
var str = c.ModifyLine(1, "Modified line");
```

Next()

Description

Returns the next comment in the model.

Arguments

No arguments

Returns

Comment object (or null if there are no more comments in the model).

Return type

Comment

Example

To get the comment in model m after comment c:

```
var c = c.Next();
```

Previous()

Description

Returns the previous comment in the model.

Arguments

No arguments

Returns

Comment object (or null if there are no more comments in the model).

Return type

Comment

Example

To get the comment in model m before comment c:

```
var c = c.Previous();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select comments using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting comments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only comments from that model can be selected. If the argument is a Flag then only comments that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any comments can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of comments selected or null if menu cancelled

Return type

Number

Example

To select comments from model m, flagging those selected with flag f, giving the prompt 'Select comments':

```
Comment.Select(f, 'Select comments', m);
```

To select comments, flagging those selected with flag f but limiting selection to comments flagged with flag l, giving the prompt 'Select comments':

```
Comment.Select(f, 'Select comments', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the comment.

Arguments

- **flag** (Flag)

Flag to set on the comment

Returns

No return value

Example

To set flag f for comment c:

```
c.SetFlag(f);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of comments in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing comments should be counted. If false or omitted referenced but undefined comments will also be included in the total.

Returns

number of comments

Return type

Number

Example

To get the total number of comments in model m:

```
var total = Comment.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the comments in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all comments will be unset in

- **flag** (Flag)

Flag to unset on the comments

Returns

No return value

Example

To unset the flag f on all the comments in model m:

```
Comment.UnflagAll(m, f);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

Comment object.

Return type

Comment

Example

To check if Comment property c.example is a parameter by using the Comment.GetParameter() method:

```
if (c.ViewParameters().GetParameter(c.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for comment with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The comment object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for comment c:

```
c.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for comment c:

```
c.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for comment. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for comment c:

```
c.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this comment.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for comment c:

```
var xrefs = c.Xrefs();
```

---

toString()

Description

Creates a string containing the comment data in keyword format. Note that this contains the keyword header and the keyword cards. See also Comment.Keyword() and Comment.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for comment c in keyword format

```
var s = c.toString();
```

ExtraNodes class

The ExtraNodes class gives you access to constrained extra nodes cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- ExtractColour()
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

ExtraNodes constants

Name	Description
ExtraNodes.NODE	CNST is *CONSTRAINED_EXTRA_NODES_NODE.
ExtraNodes.SET	CNST is *CONSTRAINED_EXTRA_NODES_SET.

ExtraNodes properties

Name	Type	Description
colour	Colour	The colour of the extra nodes
exists (read only)	logical	true if constrained extra node exists, false if referred to but not defined.
id	integer	Node ID or node set ID (not internal label)
iflag	logical	Flag for adding node mass inertia to PART_INERTIA
include	integer	The Include file number that the constrained extra node is in.
label (read only)	integer	The label the constrained extra nodes has in PRIMER
model (read only)	integer	The Model number that the constrained extra node is in.
option	constant	The Constrained Extra Nodes option. Can be ExtraNodes.NODE or ExtraNodes.SET.
pid	integer	Part ID of rigid body.

Detailed Description

The ExtraNodes class allows you to create, modify, edit and manipulate constrained extra nodes cards. See the documentation below for more details.

Constructor

new ExtraNodes(Model*[Model]*, option*[constant]*, pid*[integer]*, id*[integer]*, iflag*[boolean]*)

Description

Create a new ExtraNodes object.

Arguments

- **Model** (Model)

Model that constrained extra nodes will be created in

- **option** (constant)

Specify the type of constrained extra nodes. Can be ExtraNodes.NODE or ExtraNodes.SET)

- **pid** (integer)

Part ID of rigid body

- **id** (integer)

Node node ID or node set ID

- **iflag** (boolean)

Flag for adding node mass inertia to PART_INERTIA

Returns

ExtraNodes object

Return type

ExtraNodes

Example

To create a new constrained extra nodes in model m, of type SET, with part 9, node set 18 and iflag 0

```
var e = new ExtraNodes(m, ExtraNodes.SET, 9, 18, 0);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a constrained extra node.

Arguments

- **Comment** (Comment)

Comment that will be attached to the constrained extra node

Returns

No return value

Example

To associate comment c to the constrained extra node en:

```
en.AssociateComment(c);
```

Blank()

Description

Blanks the constrained extra node

Arguments

No arguments

Returns

No return value

Example

To blank constrained extra node en:

```
en.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the constrained extra nodes in the model.

Arguments

- **Model** (Model)

Model that all constrained extra nodes will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the constrained extra nodes in model m:

```
ExtraNodes.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged constrained extra nodes in the model.

Arguments

- **Model** (Model)

Model that all the flagged constrained extra nodes will be blanked in

- **flag** (Flag)

Flag set on the constrained extra nodes that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the constrained extra nodes in model m flagged with f:

```
ExtraNodes.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the constrained extra node is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if constrained extra node en is blanked:

```
if (en.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse constrained extra node en:

```
en.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the constrained extra node.

Arguments

- **flag** (Flag)

Flag to clear on the constrained extra node

Returns

No return value

Example

To clear flag f for constrained extra node en:

```
en.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the constrained extra node. The target include of the copied constrained extra node can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

ExtraNodes object

Return type

ExtraNodes

Example

To copy constrained extra node en into constrained extra node z:

```
var z = en.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a constrained extra node

Arguments

- **Model** (Model)

Model that the constrained extra node will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

ExtraNodes object (or null if not made).

Return type

ExtraNodes

Example

To start creating a constrained extra node in model m:

```
var en = ExtraNodes.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a constrained extra node.

Arguments

- **Comment** (Comment)

Comment that will be detached from the constrained extra node

Returns

No return value

Example

To detach comment c from the constrained extra node en:

```
en.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit constrained extra node en:

```
en.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for constrained extra node with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The constrained extra node object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for constrained extra node en:

```
en.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for constrained extra node en:

```
en.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for constrained extra node. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for constrained extra node en:

```
en.Error("My custom error");
```

ExtractColour()

Description

Extracts the **actual** colour used for constrained extra node.
By default in PRIMER many entities such as elements get their colour automatically from the part that they are in. PRIMER cycles through 13 default colours based on the label of the entity. In this case the constrained extra node colour property will return the value Colour.PART instead of the actual colour. This method will return the actual colour which is used for drawing the constrained extra node.

Arguments

No arguments

Returns

colour value (integer)

Return type

Number

Example

To return the colour used for drawing constrained extra node en:

```
var colour = en.ExtractColour();
```

First(Model*[Model]*) [static]

Description

Returns the first constrained extra node in the model.

Arguments

- **Model** (Model)

Model to get first constrained extra node in

Returns

ExtraNodes object (or null if there are no constrained extra nodes in the model).

Return type

ExtraNodes

Example

To get the first constrained extra node in model m:

```
var en = ExtraNodes.First(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the constrained extra nodes in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all constrained extra nodes will be flagged in

- **flag** (Flag)

Flag to set on the constrained extra nodes

Returns

No return value

Example

To flag all of the constrained extra nodes with flag f in model m:

```
ExtraNodes.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the constrained extra node is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the constrained extra node

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if constrained extra node en has flag f set on it:

```
if (en.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each constrained extra node in the model.
Note that ForEach has been designed to make looping over constrained extra nodes as fast as possible and so has some limitations.
Firstly, a single temporary ExtraNodes object is created and on each function call it is updated with the current constrained extra node data. This means that you should not try to store the ExtraNodes object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new constrained extra nodes inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all constrained extra nodes are in

- **func** (function)

Function to call for each constrained extra node

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the constrained extra nodes in model m:

```
ExtraNodes.ForEach(m, test);
function test(en)
{
// en is ExtraNodes object
}
```

To call function test for all of the constrained extra nodes in model m with optional object:

```
var data = { x:0, y:0 };
ExtraNodes.ForEach(m, test, data);
function test(en, extra)
{
// en is ExtraNodes object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of ExtraNodes objects or properties for all of the constrained extra nodes in a model in PRIMER. If the optional property argument is not given then an array of ExtraNodes objects is returned. If the property argument is given, that property value for each constrained extra node is returned in the array instead of a ExtraNodes object

Arguments

- **Model** (Model)

Model to get constrained extra nodes from

- **property (optional)** (string)

Name for property to get for all constrained extra nodes in the model

Returns

Array of ExtraNodes objects or properties

Return type

Array

Example

To make an array of ExtraNodes objects for all of the constrained extra nodes in model m:

```
var a = ExtraNodes.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each constrained extra node in model m:

```
var a = ExtraNodes.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a constrained extra node.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the constrained extra node en:

```
var comm_array = en.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of ExtraNodes objects for all of the flagged constrained extra nodes in a model in PRIMER If the optional property argument is not given then an array of ExtraNodes objects is returned. If the property argument is given, then that property value for each constrained extra node is returned in the array instead of a ExtraNodes object

Arguments

- **Model** (Model)

Model to get constrained extra nodes from

- **flag** (Flag)

Flag set on the constrained extra nodes that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged constrained extra nodes in the model

Returns

Array of ExtraNodes objects or properties

Return type

Array

Example

To make an array of ExtraNodes objects for all of the constrained extra nodes in model m flagged with f:

```
var en = ExtraNodes.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the constrained extra nodes in model m flagged with f:

```
var a = ExtraNodes.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the ExtraNodes object for a constrained extra node ID.

Arguments

- **Model** (Model)

Model to find the constrained extra node in

- **number** (integer)

number of the constrained extra node you want the ExtraNodes object for

Returns

ExtraNodes object (or null if constrained extra node does not exist).

Return type

ExtraNodes

Example

To get the ExtraNodes object for constrained extra node 100 in model m

```
var en = ExtraNodes.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a ExtraNodes property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the ExtraNodes.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

constrained extra node property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if ExtraNodes property en.example is a parameter:

```
Options.property_parameter_names = true;
if (en.GetParameter(en.example) ) do_something...
Options.property_parameter_names = false;
```

To check if ExtraNodes property en.example is a parameter by using the GetParameter method:

```
if (en.ViewParameters().GetParameter(en.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this constrained extra nodes (*CONSTRAINED_EXTRA_NODES). **Note that a carriage return is not added**. See also ExtraNodes.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for constrained extra nodes e:

```
var key = e.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the constrained extra nodes. **Note that a carriage return is not added**. See also ExtraNodes.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for constrained extra nodes e:

```
var cards = e.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last constrained extra node in the model.

Arguments

- **Model** (Model)

Model to get last constrained extra node in

Returns

ExtraNodes object (or null if there are no constrained extra nodes in the model).

Return type

ExtraNodes

Example

To get the last constrained extra node in model m:

```
var en = ExtraNodes.Last(m);
```

Next()

Description

Returns the next constrained extra node in the model.

Arguments

No arguments

Returns

ExtraNodes object (or null if there are no more constrained extra nodes in the model).

Return type

ExtraNodes

Example

To get the constrained extra node in model m after constrained extra node en:

```
var en = en.Next();
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a constrained extra node.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only constrained extra nodes from that model can be picked. If the argument is a Flag then only constrained extra nodes that are flagged with *limit* can be selected. If omitted, or null, any constrained extra nodes from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

ExtraNodes object (or null if not picked)

Return type

ExtraNodes

Example

To pick a constrained extra node from model m giving the prompt 'Pick constrained extra node from screen':

```
var en = ExtraNodes.Pick('Pick constrained extra node from screen', m);
```

Previous()

Description

Returns the previous constrained extra node in the model.

Arguments

No arguments

Returns

ExtraNodes object (or null if there are no more constrained extra nodes in the model).

Return type

ExtraNodes

Example

To get the constrained extra node in model m before constrained extra node en:

```
var en = en.Previous();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select constrained extra nodes using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting constrained extra nodes

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only constrained extra nodes from that model can be selected. If the argument is a Flag then only constrained extra nodes that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any constrained extra nodes can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of constrained extra nodes selected or null if menu cancelled

Return type

Number

Example

To select constrained extra nodes from model m, flagging those selected with flag f, giving the prompt 'Select constrained extra nodes':

```
ExtraNodes.Select(f, 'Select constrained extra nodes', m);
```

To select constrained extra nodes, flagging those selected with flag f but limiting selection to constrained extra nodes flagged with flag l, giving the prompt 'Select constrained extra nodes':

```
ExtraNodes.Select(f, 'Select constrained extra nodes', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the constrained extra node.

Arguments

- **flag** (Flag)

Flag to set on the constrained extra node

Returns

No return value

Example

To set flag f for constrained extra node en:

```
en.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the constrained extra node. The constrained extra node will be sketched until you either call ExtraNodes.Unsketch(), ExtraNodes.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the constrained extra node is sketched. If omitted redraw is true. If you want to sketch several constrained extra nodes and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch constrained extra node en:

```
en.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged constrained extra nodes in the model. The constrained extra nodes will be sketched until you either call ExtraNodes.Unsketch(), ExtraNodes.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged constrained extra nodes will be sketched in

- **flag** (Flag)

Flag set on the constrained extra nodes that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the constrained extra nodes are sketched. If omitted redraw is true. If you want to sketch flagged constrained extra nodes several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all constrained extra nodes flagged with flag in model m:

```
ExtraNodes.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of constrained extra nodes in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing constrained extra nodes should be counted. If false or omitted referenced but undefined constrained extra nodes will also be included in the total.

Returns

number of constrained extra nodes

Return type

Number

Example

To get the total number of constrained extra nodes in model m:

```
var total = ExtraNodes.Total(m);
```

Unblank()

Description

Unblanks the constrained extra node

Arguments

No arguments

Returns

No return value

Example

To unblank constrained extra node en:

```
en.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the constrained extra nodes in the model.

Arguments

- **Model** (Model)

Model that all constrained extra nodes will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the constrained extra nodes in model m:

```
ExtraNodes.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged constrained extra nodes in the model.

Arguments

- **Model** (Model)

Model that the flagged constrained extra nodes will be unblanked in

- **flag** (Flag)

Flag set on the constrained extra nodes that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the constrained extra nodes in model m flagged with f:

```
ExtraNodes.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the constrained extra nodes in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all constrained extra nodes will be unset in

- **flag** (Flag)

Flag to unset on the constrained extra nodes

Returns

No return value

Example

To unset the flag f on all the constrained extra nodes in model m:

```
ExtraNodes.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the constrained extra node.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the constrained extra node is unsketched. If omitted redraw is true. If you want to unsketch several constrained extra nodes and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch constrained extra node en:

```
en.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all constrained extra nodes.

Arguments

- **Model** (Model)

Model that all constrained extra nodes will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the constrained extra nodes are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all constrained extra nodes in model m:

```
ExtraNodes.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged constrained extra nodes in the model.

Arguments

- **Model** (Model)

Model that all constrained extra nodes will be unsketched in

- **flag** (Flag)

Flag set on the constrained extra nodes that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the constrained extra nodes are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all constrained extra nodes flagged with flag in model m:

```
ExtraNodes.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

ExtraNodes object.

Return type

ExtraNodes

Example

To check if ExtraNodes property en.example is a parameter by using the ExtraNodes.GetParameter() method:

```
if (en.ViewParameters().GetParameter(en.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for constrained extra node with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The constrained extra node object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...

autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for constrained extra node en:

```
en.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for constrained extra node en:

```
en.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for constrained extra node. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for constrained extra node en:

```
en.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this constrained extra node.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for constrained extra node en:

```
var xrefs = en.Xrefs();
```

---

toString()

Description

Creates a string containing the constrained extra nodes data in keyword format. Note that this contains the keyword header and the keyword cards. See also ExtraNodes.Keyword() and ExtraNodes.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for constrained extra nodes e in keyword format

```
var s = e.toString();
```

GeneralizedWeld (Gwld) class

The GeneralizedWeld class gives you access to constrained generalized weld cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model *[Model]*, redraw (optional) *[boolean]*)
- BlankFlagged(Model *[Model]*, flag *[Flag]*, redraw (optional) *[boolean]*)
- Create(Model *[Model]*, modal (optional) *[boolean]*)
- First(Model *[Model]*)
- FirstFreeLabel(Model *[Model]*, layer (optional) *[Include number]*)
- FlagAll(Model *[Model]*, flag *[Flag]*)
- ForEach(Model *[Model]*, func *[function]*, extra (optional) *[any]*)
- GetAll(Model *[Model]*, property (optional) *[string]*)
- GetFlagged(Model *[Model]*, flag *[Flag]*, property (optional) *[string]*)
- GetFromID(Model *[Model]*, number *[integer]*)
- Last(Model *[Model]*)
- LastFreeLabel(Model *[Model]*, layer (optional) *[Include number]*)
- NextFreeLabel(Model *[Model]*, layer (optional) *[Include number]*)
- Pick(prompt *[string]*, limit (optional) *[Model or Flag]*, modal (optional) *[boolean]*, button text (optional) *[string]*)
- RenumberAll(Model *[Model]*, start *[integer]*)
- RenumberFlagged(Model *[Model]*, flag *[Flag]*, start *[integer]*)
- Select(flag *[Flag]*, prompt *[string]*, limit (optional) *[Model or Flag]*, modal (optional) *[boolean]*)
- SketchFlagged(Model *[Model]*, flag *[Flag]*, redraw (optional) *[boolean]*)
- Total(Model *[Model]*, exists (optional) *[boolean]*)
- UnblankAll(Model *[Model]*, redraw (optional) *[boolean]*)
- UnblankFlagged(Model *[Model]*, flag *[Flag]*, redraw (optional) *[boolean]*)
- UnflagAll(Model *[Model]*, flag *[Flag]*)
- UnsketchAll(Model *[Model]*, redraw (optional) *[boolean]*)
- UnsketchFlagged(Model *[Model]*, flag *[Flag]*, redraw (optional) *[boolean]*)

Member functions

- AssociateComment(Comment *[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional) *[boolean]*)
- ClearFlag(flag *[Flag]*)
- Copy(range (optional) *[boolean]*)
- DetachComment(Comment *[Comment]*)
- Edit(modal (optional) *[boolean]*)
- Error(message *[string]*, options (optional) *[object]*)
- Error(message *[string]*, details (optional) *[string]*) [deprecated]
- Flagged(flag *[Flag]*)
- GetCombinedData(index *[integer]*)
- GetComments()
- GetCrossFilletData(index *[integer]*)
- GetParameter(prop *[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetCombinedData(index *[integer]*, data *[Array of numbers]*)
- SetCrossFilletData(index *[integer]*, data *[Array of numbers]*)
- SetFlag(flag *[Flag]*)
- Sketch(redraw (optional) *[boolean]*)
- Unblank()

- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

GeneralizedWeld constants

Name	Description
GeneralizedWeld.BUTT	GeneralizedWeld is *CONSTRAINED_GENERALIZED_WELD_BUTT.
GeneralizedWeld.COMBINED	GeneralizedWeld is *CONSTRAINED_GENERALIZED_WELD_COMBINED.
GeneralizedWeld.CROSS_FILLET	GeneralizedWeld is *CONSTRAINED_GENERALIZED_WELD_CROSS_FILLET.
GeneralizedWeld.FILLET	GeneralizedWeld is *CONSTRAINED_GENERALIZED_WELD_FILLET.
GeneralizedWeld.SPOT	GeneralizedWeld is *CONSTRAINED_GENERALIZED_WELD_SPOT.

GeneralizedWeld properties

Name	Type	Description
a	real	Width of fillet (GeneralizedWeld.FILLET, GeneralizedWeld.CROSS_FILLET)
alpha	real	Weld angle (GeneralizedWeld.FILLET, GeneralizedWeld.CROSS_FILLET)
beta	real	Failure parameter (GeneralizedWeld.FILLET, GeneralizedWeld.BUTT, GeneralizedWeld.CROSS_FILLET)
cid	integer	Coordinate System ID.
d	real	Thickness of weld (GeneralizedWeld.BUTT)
epsf	real	Effective plastic strain at failure (GeneralizedWeld.SPOT, GeneralizedWeld.FILLET, GeneralizedWeld.BUTT, GeneralizedWeld.CROSS_FILLET)
exists (read only)	logical	true if generalized weld exists, false if referred to but not defined.
filter	integer	Number of force vectors saved for filtering.
id	logical	true if _ID option is set, false if not
include	integer	The Include file number that the generalized weld is in.
l	real	Length of weld (GeneralizedWeld.FILLET, GeneralizedWeld.BUTT, GeneralizedWeld.CROSS_FILLET)
label	integer	Constrained Generalized weld number.
lt	real	Transverse length (GeneralizedWeld.BUTT)
m	real	Exponent for shear force (GeneralizedWeld.SPOT)
model (read only)	integer	The Model number that the generalized weld is in.
n	real	Exponent for normal force (GeneralizedWeld.SPOT)
npr	integer	Number of individual nodal pairs in cross fillet and combined weld.
nprt	integer	Printout option.
nsid	integer	Set Node Set ID.
option	constant	GeneralizedWeld type. Can be GeneralizedWeld.SPOT, GeneralizedWeld.FILLET, GeneralizedWeld.BUTT, GeneralizedWeld.CROSS_FILLET, GeneralizedWeld.COMBINED

sigf	real	Stress at failure (GeneralizedWeld.FILLET)
sigy	real	Stress at failure (GeneralizedWeld.BUTT, GeneralizedWeld.CROSS_FILLET)
sn	real	Normal force at failure (GeneralizedWeld.SPOT)
ss	real	Shear force at faliure (GeneralizedWeld.SPOT)
tfail	real	Failure time for constraint set (GeneralizedWeld.SPOT, GeneralizedWeld.FILLET, GeneralizedWeld.BUTT, GeneralizedWeld.CROSS_FILLET)
w	real	Width of flange (GeneralizedWeld.FILLET, GeneralizedWeld.CROSS_FILLET)
wid	integer	Constrained Generalized weld number (identical to label).
window	real	Filter time window.

Detailed Description

The GeneralizedWeld class allows you to create, modify, edit and manipulate generalized weld cards. See the documentation below for more details.

For convenience "Gwld" can also be used as the class name instead of "GeneralizedWeld".

Constructor

new GeneralizedWeld(Model*[Model]*, option*[constant]*, nsid*[integer]*, cid (optional)*[integer]*, filter (optional)*[integer]*, window (optional)*[real]*, npr (optional)*[integer]*, nprt (optional)*[integer]*, wid (optional)*[integer]*)

Description

Create a new GeneralizedWeld object.

Arguments

- **Model** (Model)

Model that gwld will be created in

- **option** (constant)

Constrained generalized weld type (any).

- **nsid** (integer)

Set Node Set ID.

- **cid (optional)** (integer)

Coordinate System ID.

- **filter (optional)** (integer)

Number of force vectors saved for filtering.

- **window (optional)** (real)

Filter time window.

- **npr (optional)** (integer)

Number of individual nodal pairs in cross fillet and combined weld.

- **nprt (optional)** (integer)

Printout option.

- **wid (optional)** (integer)

Constrained Generalized weld number.

Returns

GeneralizedWeld object

Return type

GeneralizedWeld

Example

To create a new gwld 1000 of type SPOT in model m with specifiction: nsid, cid, filter, window, nprt are 91, 92, 81, 0.5, 82 respectively

```
var w = new GeneralizedWeld(m, GeneralizedWeld.SPOT, 91, 92, 81, 0.5, 82, 1000);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a generalized weld.

Arguments

- **Comment** (Comment)

Comment that will be attached to the generalized weld

Returns

No return value

Example

To associate comment c to the generalized weld gw:

```
gw.AssociateComment(c);
```

Blank()

Description

Blanks the generalized weld

Arguments

No arguments

Returns

No return value

Example

To blank generalized weld gw:

```
gw.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the generalized welds in the model.

Arguments

- **Model** (Model)

Model that all generalized welds will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the generalized welds in model m:

```
GeneralizedWeld.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged generalized welds in the model.

Arguments

- **Model** (Model)

Model that all the flagged generalized welds will be blanked in

- **flag** (Flag)

Flag set on the generalized welds that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the generalized welds in model m flagged with f:

```
GeneralizedWeld.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the generalized weld is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if generalized weld gw is blanked:

```
if (gw.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse generalized weld gw:

```
gw.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the generalized weld.

Arguments

- **flag** (Flag)

Flag to clear on the generalized weld

Returns

No return value

Example

To clear flag f for generalized weld gw:

```
gw.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the generalized weld. The target include of the copied generalized weld can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

GeneralizedWeld object

Return type

GeneralizedWeld

Example

To copy generalized weld gw into generalized weld z:

```
var z = gw.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a generalized weld

Arguments

- **Model** (Model)

Model that the generalized weld will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

GeneralizedWeld object (or null if not made).

Return type

GeneralizedWeld

Example

To start creating a generalized weld in model m:

```
var gw = GeneralizedWeld.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a generalized weld.

Arguments

- **Comment** (Comment)

Comment that will be detached from the generalized weld

Returns

No return value

Example

To detach comment c from the generalized weld gw:

```
gw.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit generalized weld gw:

```
gw.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for generalized weld with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The generalized weld object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for generalized weld gw:

```
gw.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for generalized weld gw:

```
gw.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for generalized weld. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for generalized weld gw:

```
gw.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first generalized weld in the model.

Arguments

- **Model** (Model)

Model to get first generalized weld in

Returns

GeneralizedWeld object (or null if there are no generalized welds in the model).

Return type

GeneralizedWeld

Example

To get the first generalized weld in model m:

```
var gw = GeneralizedWeld.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free generalized weld label in the model. Also see GeneralizedWeld.LastFreeLabel(), GeneralizedWeld.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free generalized weld label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

GeneralizedWeld label.

Return type

Number

Example

To get the first free generalized weld label in model m:

```
var label = GeneralizedWeld.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the generalized welds in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all generalized welds will be flagged in

- **flag** (Flag)

Flag to set on the generalized welds

Returns

No return value

Example

To flag all of the generalized welds with flag f in model m:

```
GeneralizedWeld.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the generalized weld is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the generalized weld

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if generalized weld gw has flag f set on it:

```
if (gw.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each generalized weld in the model.
Note that ForEach has been designed to make looping over generalized welds as fast as possible and so has some limitations.
Firstly, a single temporary GeneralizedWeld object is created and on each function call it is updated with the current generalized weld data. This means that you should not try to store the GeneralizedWeld object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new generalized welds inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all generalized welds are in

- **func** (function)

Function to call for each generalized weld

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the generalized welds in model m:

```
GeneralizedWeld.ForEach(m, test);
function test(gw)
{
// gw is GeneralizedWeld object
}
```

To call function test for all of the generalized welds in model m with optional object:

```
var data = { x:0, y:0 };
GeneralizedWeld.ForEach(m, test, data);
function test(gw, extra)
{
// gw is GeneralizedWeld object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of GeneralizedWeld objects or properties for all of the generalized welds in a model in PRIMER. If the optional property argument is not given then an array of GeneralizedWeld objects is returned. If the property argument is given, that property value for each generalized weld is returned in the array instead of a GeneralizedWeld object

Arguments

- **Model** (Model)

Model to get generalized welds from

- **property (optional)** (string)

Name for property to get for all generalized welds in the model

Returns

Array of GeneralizedWeld objects or properties

Return type

Array

Example

To make an array of GeneralizedWeld objects for all of the generalized welds in model m:

```
var a = GeneralizedWeld.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each generalized weld in model m:

```
var a = GeneralizedWeld.GetAll(m, 'foo');
```

GetCombinedData(index*[integer]*)

Description

Returns the combined data for a specific nodal pair as an array.

Arguments

- **index** (integer)

Index you want the data for. **Note that indices start at 0**.

Returns

An array containing the data (tfail, epsf, sigy, beta, l, w, a, alpha, nodea, nodeb, ncid, wtyp).

Return type

Array

Example

To get the data for the 3rd node pair for generalized weld gw:

```
var data = gw.GetCombinedData(2);
```

GetComments()

Description

Extracts the comments associated to a generalized weld.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the generalized weld gw:

```
var comm_array = gw.GetComments();
```

GetCrossFilletData(index*[integer]*)

Description

Returns the cross fillet data for a specific nodal pair as an array.

Arguments

- **index** (integer)

Index you want the data for. **Note that indices start at 0**.

Returns

An array containing the data (nodea, nodeb, ncid).

Return type

Array

Example

To get the data for the 3rd node pair for generalized weld gw:

```
var data = gw.GetCrossFilletData(2);
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of GeneralizedWeld objects for all of the flagged generalized welds in a model in PRIMER If the optional property argument is not given then an array of GeneralizedWeld objects is returned. If the property argument is given, then that property value for each generalized weld is returned in the array instead of a GeneralizedWeld object

Arguments

- **Model** (Model)

Model to get generalized welds from

- **flag** (Flag)

Flag set on the generalized welds that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged generalized welds in the model

Returns

Array of GeneralizedWeld objects or properties

Return type

Array

Example

To make an array of GeneralizedWeld objects for all of the generalized welds in model m flagged with f:

```
var gw = GeneralizedWeld.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the generalized welds in model m flagged with f:

```
var a = GeneralizedWeld.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the GeneralizedWeld object for a generalized weld ID.

Arguments

- **Model** (Model)

Model to find the generalized weld in

- **number** (integer)

number of the generalized weld you want the GeneralizedWeld object for

Returns

GeneralizedWeld object (or null if generalized weld does not exist).

Return type

GeneralizedWeld

Example

To get the GeneralizedWeld object for generalized weld 100 in model m

```
var gw = GeneralizedWeld.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a GeneralizedWeld property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the GeneralizedWeld.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

generalized weld property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if GeneralizedWeld property gw.example is a parameter:

```
Options.property_parameter_names = true;
if (gw.GetParameter(gw.example) ) do_something...
Options.property_parameter_names = false;
```

To check if GeneralizedWeld property gw.example is a parameter by using the GetParameter method:

```
if (gw.ViewParameters().GetParameter(gw.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this gwld (*CONSTRAINED_GENERALIZED_WELD_xxxx). **Note that a carriage return is not added**. See also GeneralizedWeld.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for generalized weld gw:

```
var key = gw.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the gwld. **Note that a carriage return is not added**. See also GeneralizedWeld.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for generalized weld gw:

```
var cards = gw.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last generalized weld in the model.

Arguments

- **Model** (Model)

Model to get last generalized weld in

Returns

GeneralizedWeld object (or null if there are no generalized welds in the model).

Return type

GeneralizedWeld

Example

To get the last generalized weld in model m:

```
var gw = GeneralizedWeld.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free generalized weld label in the model. Also see GeneralizedWeld.FirstFreeLabel(), GeneralizedWeld.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free generalized weld label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

GeneralizedWeld label.

Return type

Number

Example

To get the last free generalized weld label in model m:

```
var label = GeneralizedWeld.LastFreeLabel(m);
```

Next()

Description

Returns the next generalized weld in the model.

Arguments

No arguments

Returns

GeneralizedWeld object (or null if there are no more generalized welds in the model).

Return type

GeneralizedWeld

Example

To get the generalized weld in model m after generalized weld gw:

```
var gw = gw.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) generalized weld label in the model. Also see GeneralizedWeld.FirstFreeLabel(), GeneralizedWeld.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free generalized weld label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

GeneralizedWeld label.

Return type

Number

Example

To get the next free generalized weld label in model m:

```
var label = GeneralizedWeld.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a generalized weld.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only generalized welds from that model can be picked. If the argument is a Flag then only generalized welds that are flagged with *limit* can be selected. If omitted, or null, any generalized welds from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

GeneralizedWeld object (or null if not picked)

Return type

GeneralizedWeld

Example

To pick a generalized weld from model m giving the prompt 'Pick generalized weld from screen':

```
var gw = GeneralizedWeld.Pick('Pick generalized weld from screen', m);
```

Previous()

Description

Returns the previous generalized weld in the model.

Arguments

No arguments

Returns

GeneralizedWeld object (or null if there are no more generalized welds in the model).

Return type

GeneralizedWeld

Example

To get the generalized weld in model m before generalized weld gw:

```
var gw = gw.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the generalized welds in the model.

Arguments

- **Model** (Model)

Model that all generalized welds will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the generalized welds in model m, from 1000000:

```
GeneralizedWeld.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged generalized welds in the model.

Arguments

- **Model** (Model)

Model that all the flagged generalized welds will be renumbered in

- **flag** (Flag)

Flag set on the generalized welds that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the generalized welds in model m flagged with f, from 1000000:

```
GeneralizedWeld.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select generalized welds using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting generalized welds

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only generalized welds from that model can be selected. If the argument is a Flag then only generalized welds that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any generalized welds can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of generalized welds selected or null if menu cancelled

Return type

Number

Example

To select generalized welds from model m, flagging those selected with flag f, giving the prompt 'Select generalized welds':

```
GeneralizedWeld.Select(f, 'Select generalized welds', m);
```

To select generalized welds, flagging those selected with flag f but limiting selection to generalized welds flagged with flag l, giving the prompt 'Select generalized welds':

```
GeneralizedWeld.Select(f, 'Select generalized welds', l);
```

SetCombinedData(index*[integer]*, data*[Array of numbers]*)

Description

Sets the combined data for a specific nodal pair.

Arguments

- **index** (integer)

Index you want to set the data for. **Note that indices start at 0**.

- **data** (Array of numbers)

Array containing the data. The array length should be 12 (tfail, epsf, sigy, beta, l, w, a, alpha, nodea, nodeb, ncid, wtyp)

Returns

No return value.

Example

To set the data for the 3rd nodal pair for generalized weld gw to the values in array adata:

```
gw.SetCombinedData(2, adata);
```

SetCrossFilletData(index*[integer]*, data*[Array of numbers]*)

Description

Sets the cross fillet data for a specific nodal pair.

Arguments

- **index** (integer)

Index you want to set the data for. **Note that indices start at 0**.

- **data** (Array of numbers)

Array containing the data. The array length should be 3 (nodea, nodeb, ncid)

Returns

No return value.

Example

To set the data for the 3rd nodal pair for generalized weld gw to the values in array adata:

```
gw.SetCrossFilletData(2, adata);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the generalized weld.

Arguments

- **flag** (Flag)

Flag to set on the generalized weld

Returns

No return value

Example

To set flag f for generalized weld gw:

```
gw.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the generalized weld. The generalized weld will be sketched until you either call GeneralizedWeld.Unsketch(), GeneralizedWeld.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the generalized weld is sketched. If omitted redraw is true. If you want to sketch several generalized welds and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch generalized weld gw:

```
gw.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged generalized welds in the model. The generalized welds will be sketched until you either call GeneralizedWeld.Unsketch(), GeneralizedWeld.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged generalized welds will be sketched in

- **flag** (Flag)

Flag set on the generalized welds that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the generalized welds are sketched. If omitted redraw is true. If you want to sketch flagged generalized welds several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all generalized welds flagged with flag in model m:

```
GeneralizedWeld.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of generalized welds in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing generalized welds should be counted. If false or omitted referenced but undefined generalized welds will also be included in the total.

Returns

number of generalized welds

Return type

Number

Example

To get the total number of generalized welds in model m:

```
var total = GeneralizedWeld.Total(m);
```

Unblank()

Description

Unblanks the generalized weld

Arguments

No arguments

Returns

No return value

Example

To unblank generalized weld gw:

```
gw.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the generalized welds in the model.

Arguments

- **Model** (Model)

Model that all generalized welds will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the generalized welds in model m:

```
GeneralizedWeld.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged generalized welds in the model.

Arguments

- **Model** (Model)

Model that the flagged generalized welds will be unblanked in

- **flag** (Flag)

Flag set on the generalized welds that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the generalized welds in model m flagged with f:

```
GeneralizedWeld.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the generalized welds in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all generalized welds will be unset in

- **flag** (Flag)

Flag to unset on the generalized welds

Returns

No return value

Example

To unset the flag f on all the generalized welds in model m:

```
GeneralizedWeld.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the generalized weld.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the generalized weld is unsketched. If omitted redraw is true. If you want to unsketch several generalized welds and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch generalized weld gw:

```
gw.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all generalized welds.

Arguments

- **Model** (Model)

Model that all generalized welds will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the generalized welds are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all generalized welds in model m:

```
GeneralizedWeld.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged generalized welds in the model.

Arguments

- **Model** (Model)

Model that all generalized welds will be unsketched in

- **flag** (Flag)

Flag set on the generalized welds that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the generalized welds are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all generalized welds flagged with flag in model m:

```
GeneralizedWeld.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

GeneralizedWeld object.

Return type

GeneralizedWeld

Example

To check if GeneralizedWeld property gw.example is a parameter by using the GeneralizedWeld.GetParameter() method:

```
if (gw.ViewParameters().GetParameter(gw.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for generalized weld with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The generalized weld object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for generalized weld gw:

```
gw.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for generalized weld gw:

```
gw.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for generalized weld. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for generalized weld gw:

```
gw.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this generalized weld.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for generalized weld gw:

```
var xrefs = gw.Xrefs();
```

---

toString()

Description

Creates a string containing the gwld data in keyword format. Note that this contains the keyword header and the keyword cards. See also GeneralizedWeld.Keyword() and GeneralizedWeld.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for generalized weld gw in keyword format

```
var s = gw.toString();
```

Interpolation class

The Interpolation class gives you access to define *CONSTRAINED_INTERPOLATION cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AddRowData(inid*[integer]*, idof (optional)*[integer]*, twghtx (optional)*[real]*, twghty (optional)*[real]*, twghtz (optional)*[real]*, rwghtx (optional)*[real]*, rwghty (optional)*[real]*, rwghtz (optional)*[real]*, cidi (optional)*[integer]*)
- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- GetRowData(row_index*[Integer]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- RemoveRowData(row_index*[Integer]*)
- SetFlag(flag*[Flag]*)
- SetRowData(row_index*[Integer]*, inid*[integer]*, idof (optional)*[integer]*, twghtx (optional)*[real]*, twghty

(optional)*[real]*, twghtz (optional)*[real]*, rwghtx (optional)*[real]*, rwghty (optional)*[real]*, rwghtz (optional)*[real]*, cidi (optional)*[integer]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

Interpolation constants

Name	Description
Interpolation.NODE	INID is a node.
Interpolation.NODE_SET	INID is a node set.

Interpolation properties

Name	Type	Description
cidd	integer	Coordinate System ID if LOCAL option is active.
ddof	integer	Dependent Degrees-of-Freedom.
dnid	integer	Dependent Node id.
exists (read only)	logical	true if constrained interpolation exists, false if referred to but not defined.
fgm	integer	Flag for special treatment of this constraint for implicit problems only.
icid	integer	Interpolation label
include	integer	The Include file number that the constrained interpolation is in.
indsw	integer	Switch for controlling the explicit solution when an independent (or dependent) node is deleted.
ityp	constant	The Independent Node type. Can be Interpolation.NODE or Interpolation.NODE_SET.
local	logical	true if _LOCAL is set.
model (read only)	integer	The Model number that the constrained interpolation is in.
total (read only)	integer	Total number of INID fields in the keyword.

Detailed Description

The Interpolation class allows you to create, modify, edit and manipulate *CONSTRAINED_INTERPOLATION cards. See the documentation below for more details.

Constructor

new Interpolation(Model*[Model]*, icid*[integer]*, dnid*[integer]*, inid*[integer]*, ddof (optional)*[integer]*, local (optional)*[boolean]*, cidd (optional)*[integer]*, ityp (optional)*[constant]*, idof (optional)*[integer]*, twghtx (optional)*[real]*, twghty (optional)*[real]*, twghtz (optional)*[real]*, rwghtx (optional)*[real]*, rwghty (optional)*[real]*, rwghtz (optional)*[real]*, cidi (optional)*[integer]*)

Description

Create a new Interpolation object.

Arguments

- **Model** (Model)

Model that Interpolation will be created in

- **icid** (integer)

Interpolation label

- **dnid** (integer)

Dependent Node id.

- **inid** (integer)

Independent Node or Node Set id.

- **ddof (optional)** (integer)

Dependent Degrees-of-Freedom. The default value is 123456.

- **local (optional)** (boolean)

true if _LOCAL is set.

- **cidd (optional)** (integer)

Coordinate System ID if LOCAL option is active. The default value is 0.

- **ityp (optional)** (constant)

The Independent Node type. Can be Interpolation.NODE or Interpolation.NODE_SET. The default value is Interpolation.NODE.

- **idof (optional)** (integer)

Independent Degrees-of-Freedom. The default value is 123456.

- **twghtx (optional)** (real)

Weighting factor for INID. Scales the x-translational component. The default value is 1.0.

- **twghty (optional)** (real)

Weighting factor for INID. Scales the y-translational component. The default value is twghtx.

- **twghtz (optional)** (real)

Weighting factor for INID. Scales the z-translational component. The default value is twghtx.

- **rwghtx (optional)** (real)

Weighting factor for INID. Scales the x-rotational component. The default value is twghtx.

- **rwghty (optional)** (real)

Weighting factor for INID. Scales the y-rotational component. The default value is twghtx.

- **rwghtz (optional)** (real)

Weighting factor for INID. Scales the z-rotational component. The default value is twghtx.

- **cidi (optional)** (integer)

Coordinate System ID if LOCAL option is active. The default value is 0

Returns

Interpolation object

Return type

Interpolation

Example

To create a new constrained interpolation in model m, of icid 2, dnid 12, inid 10, ddof 123, local true, cidd 22, ityp NODE_SET, idof 12, and twghtx 2.24.

```
var c_i = new Interpolation(m,2,12,10,123,true,22,Interpolation.NODE_
SET,12,2.24);
```

Details of functions

AddRowData(inid*[integer]*, idof (optional)*[integer]*, twghtx (optional)*[real]*, twghty (optional)*[real]*, twghtz (optional)*[real]*, rwghtx (optional)*[real]*, rwghty (optional)*[real]*, rwghtz (optional)*[real]*, cidi (optional)*[integer]*)

Description

Used to add additional independent node card and local coordinate card (if ITYP is Interpolation.NODE_SET) to the keyword. Adds this data to the end of the selected *CONSTRAINED_INTERPOLATION

Arguments

- **inid** (integer)

Independent Node or Node Set id.

- **idof (optional)** (integer)

Independent Degrees-of-Freedom. The default value is 123456.

- **twghtx (optional)** (real)

Weighting factor for INID. Scales the x-translational component. The default value is 1.0.

- **twghty (optional)** (real)

Weighting factor for INID. Scales the y-translational component. The default value is twghtx.

- **twghtz (optional)** (real)

Weighting factor for INID. Scales the z-translational component. The default value is twghtx.

- **rwghtx (optional)** (real)

Weighting factor for INID. Scales the x-rotational component. The default value is twghtx.

- **rwghty (optional)** (real)

Weighting factor for INID. Scales the y-rotational component. The default value is twghtx.

- **rwghtz (optional)** (real)

Weighting factor for INID. Scales the z-rotational component. The default value is twghtx.

- **cidi (optional)** (integer)

Coordinate System ID if LOCAL option is active. The default value is 0.

Returns

No return value

Example

To add INID 10 to the keyword c_i with idof 123, twghtx 1.2, twghty 2.2:

```
c_i.AddRowData(10,123,1.2,2.2);
```

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a constrained interpolation.

Arguments

- **Comment** (Comment)

Comment that will be attached to the constrained interpolation

Returns

No return value

Example

To associate comment c to the constrained interpolation c_i:

```
c_i.AssociateComment(c);
```

Blank()

Description

Blanks the constrained interpolation

Arguments

No arguments

Returns

No return value

Example

To blank constrained interpolation c_i:

```
c_i.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the constrained interpolations in the model.

Arguments

- **Model** (Model)

Model that all constrained interpolations will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the constrained interpolations in model m:

```
Interpolation.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged constrained interpolations in the model.

Arguments

- **Model** (Model)

Model that all the flagged constrained interpolations will be blanked in

- **flag** (Flag)

Flag set on the constrained interpolations that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the constrained interpolations in model m flagged with f:

```
Interpolation.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the constrained interpolation is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if constrained interpolation c_i is blanked:

```
if (c_i.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse constrained interpolation c_i:

```
c_i.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the constrained interpolation.

Arguments

- **flag** (Flag)

Flag to clear on the constrained interpolation

Returns

No return value

Example

To clear flag f for constrained interpolation c_i:

```
c_i.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the constrained interpolation. The target include of the copied constrained interpolation can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

Interpolation object

Return type

Interpolation

Example

To copy constrained interpolation c_i into constrained interpolation z:

```
var z = c_i.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a constrained interpolation

Arguments

- **Model** (Model)

Model that the constrained interpolation will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

Interpolation object (or null if not made).

Return type

Interpolation

Example

To start creating a constrained interpolation in model m:

```
var c_i = Interpolation.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a constrained interpolation.

Arguments

- **Comment** (Comment)

Comment that will be detached from the constrained interpolation

Returns

No return value

Example

To detach comment c from the constrained interpolation c_i:

```
c_i.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit constrained interpolation c_i:

```
c_i.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for constrained interpolation with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The constrained interpolation object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for constrained interpolation c_i:

```
c_i.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for constrained interpolation c_i:

```
c_i.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for constrained interpolation. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for constrained interpolation c_i:

```
c_i.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first constrained interpolation in the model.

Arguments

- **Model** (Model)

Model to get first constrained interpolation in

Returns

Interpolation object (or null if there are no constrained interpolations in the model).

Return type

Interpolation

Example

To get the first constrained interpolation in model m:

```
var c_i = Interpolation.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free constrained interpolation label in the model. Also see Interpolation.LastFreeLabel(), Interpolation.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free constrained interpolation label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

Interpolation label.

Return type

Number

Example

To get the first free constrained interpolation label in model m:

```
var label = Interpolation.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the constrained interpolations in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all constrained interpolations will be flagged in

- **flag** (Flag)

Flag to set on the constrained interpolations

Returns

No return value

Example

To flag all of the constrained interpolations with flag f in model m:

```
Interpolation.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the constrained interpolation is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the constrained interpolation

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if constrained interpolation c_i has flag f set on it:

```
if (c_i.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each constrained interpolation in the model.
Note that ForEach has been designed to make looping over constrained interpolations as fast as possible and so has some limitations.
Firstly, a single temporary Interpolation object is created and on each function call it is updated with the current constrained interpolation data. This means that you should not try to store the Interpolation object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new constrained interpolations inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all constrained interpolations are in

- **func** (function)

Function to call for each constrained interpolation

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the constrained interpolations in model m:

```
Interpolation.ForEach(m, test);
function test(c_i)
{
// c_i is Interpolation object
}
```

To call function test for all of the constrained interpolations in model m with optional object:

```
var data = { x:0, y:0 };
Interpolation.ForEach(m, test, data);
function test(c_i, extra)
{
// c_i is Interpolation object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Interpolation objects or properties for all of the constrained interpolations in a model in PRIMER. If the optional property argument is not given then an array of Interpolation objects is returned. If the property argument is given, that property value for each constrained interpolation is returned in the array instead of a Interpolation object

Arguments

- **Model** (Model)

Model to get constrained interpolations from

- **property (optional)** (string)

Name for property to get for all constrained interpolations in the model

Returns

Array of Interpolation objects or properties

Return type

Array

Example

To make an array of Interpolation objects for all of the constrained interpolations in model m:

```
var a = Interpolation.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each constrained interpolation in model m:

```
var a = Interpolation.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a constrained interpolation.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the constrained interpolation c_i:

```
var comm_array = c_i.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of Interpolation objects for all of the flagged constrained interpolations in a model in PRIMER If the optional property argument is not given then an array of Interpolation objects is returned. If the property argument is given, then that property value for each constrained interpolation is returned in the array instead of a Interpolation object

Arguments

- **Model** (Model)

Model to get constrained interpolations from

- **flag** (Flag)

Flag set on the constrained interpolations that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged constrained interpolations in the model

Returns

Array of Interpolation objects or properties

Return type

Array

Example

To make an array of Interpolation objects for all of the constrained interpolations in model m flagged with f:

```
var c_i = Interpolation.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the constrained interpolations in model m flagged with f:

```
var a = Interpolation.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the Interpolation object for a constrained interpolation ID.

Arguments

- **Model** (Model)

Model to find the constrained interpolation in

- **number** (integer)

number of the constrained interpolation you want the Interpolation object for

Returns

Interpolation object (or null if constrained interpolation does not exist).

Return type

Interpolation

Example

To get the Interpolation object for constrained interpolation 100 in model m

```
var c_i = Interpolation.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a Interpolation property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the Interpolation.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

constrained interpolation property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if Interpolation property c_i.example is a parameter:

```
Options.property_parameter_names = true;
if (c_i.GetParameter(c_i.example) ) do_something...
Options.property_parameter_names = false;
```

To check if Interpolation property c_i.example is a parameter by using the GetParameter method:

```
if (c_i.ViewParameters().GetParameter(c_i.example) ) do_something...
```

GetRowData(row_index*[Integer]*)

Description

Returns independent node cards and local coordinate cards (if ITYP is Interpolation.NODE_SET) for the selected row of the *CONSTRAINED_INTERPOLATION.

Arguments

- **row_index** (Integer)

The row index of the data to return. **Note that indices start at 0, not 1**.
0 <= row_index < Interpolation.total

Returns

Array containing data.

Return type

Array

Example

To loop over all the lines of the keyword for c_i:

```
for (i=0; i<c_i.total; i++)
        var data = c_i.GetRowData(i);
```

Keyword()

Description

Returns the keyword for this Interpolation (*constrained_interpolation). **Note that a carriage return is not added**. See also Interpolation.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for Interpolation c_i:

```
var key = c_i.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the Interpolation. **Note that a carriage return is not added**. See also Interpolation.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for Interpolation c_i:

```
var cards = c_i.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last constrained interpolation in the model.

Arguments

- **Model** (Model)

Model to get last constrained interpolation in

Returns

Interpolation object (or null if there are no constrained interpolations in the model).

Return type

Interpolation

Example

To get the last constrained interpolation in model m:

```
var c_i = Interpolation.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free constrained interpolation label in the model. Also see Interpolation.FirstFreeLabel(), Interpolation.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free constrained interpolation label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

Interpolation label.

Return type

Number

Example

To get the last free constrained interpolation label in model m:

```
var label = Interpolation.LastFreeLabel(m);
```

Next()

Description

Returns the next constrained interpolation in the model.

Arguments

No arguments

Returns

Interpolation object (or null if there are no more constrained interpolations in the model).

Return type

Interpolation

Example

To get the constrained interpolation in model m after constrained interpolation c_i:

```
var c_i = c_i.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) constrained interpolation label in the model. Also see Interpolation.FirstFreeLabel(), Interpolation.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free constrained interpolation label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

Interpolation label.

Return type

Number

Example

To get the next free constrained interpolation label in model m:

```
var label = Interpolation.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a constrained interpolation.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only constrained interpolations from that model can be picked. If the argument is a Flag then only constrained interpolations that are flagged with *limit* can be selected. If omitted, or null, any constrained interpolations from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

Interpolation object (or null if not picked)

Return type

Interpolation

Example

To pick a constrained interpolation from model m giving the prompt 'Pick constrained interpolation from screen':

```
var c_i = Interpolation.Pick('Pick constrained interpolation from screen', m);
```

Previous()

Description

Returns the previous constrained interpolation in the model.

Arguments

No arguments

Returns

Interpolation object (or null if there are no more constrained interpolations in the model).

Return type

Interpolation

Example

To get the constrained interpolation in model m before constrained interpolation c_i:

```
var c_i = c_i.Previous();
```

RemoveRowData(row_index*[Integer]*)

Description

Removes an independent node card and a local coordinate card (if ITYP is Interpolation.NODE_SET) for the selected row on the *CONSTRAINED_INTERPOLATION.

Arguments

- **row_index** (Integer)

The row index of the data to return. **Note that indices start at 0, not 1**.
0 <= row_index < Interpolation.total

Returns

No return value.

Example

To remove row 2 for c_i:

```
c_i.RemoveRowData(1);
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the constrained interpolations in the model.

Arguments

- **Model** (Model)

Model that all constrained interpolations will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the constrained interpolations in model m, from 1000000:

```
Interpolation.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged constrained interpolations in the model.

Arguments

- **Model** (Model)

Model that all the flagged constrained interpolations will be renumbered in

- **flag** (Flag)

Flag set on the constrained interpolations that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the constrained interpolations in model m flagged with f, from 1000000:

```
Interpolation.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select constrained interpolations using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting constrained interpolations

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only constrained interpolations from that model can be selected. If the argument is a Flag then only constrained interpolations that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any constrained interpolations can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of constrained interpolations selected or null if menu cancelled

Return type

Number

Example

To select constrained interpolations from model m, flagging those selected with flag f, giving the prompt 'Select constrained interpolations':

```
Interpolation.Select(f, 'Select constrained interpolations', m);
```

To select constrained interpolations, flagging those selected with flag f but limiting selection to constrained interpolations flagged with flag l, giving the prompt 'Select constrained interpolations':

```
Interpolation.Select(f, 'Select constrained interpolations', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the constrained interpolation.

Arguments

- **flag** (Flag)

Flag to set on the constrained interpolation

Returns

No return value

Example

To set flag f for constrained interpolation c_i:

```
c_i.SetFlag(f);
```

SetRowData(row_index*[Integer]*, inid*[integer]*, idof (optional)*[integer]*, twghtx (optional)*[real]*, twghty (optional)*[real]*, twghtz (optional)*[real]*, rwghtx (optional)*[real]*, rwghty (optional)*[real]*, rwghtz (optional)*[real]*, cidi (optional)*[integer]*)

Description

Used to reset values in already existing independent node cards and local coordinate cards (if ITYP is Interpolation.NODE_SET) in the selected row of *CONSTRAINED_INTERPOLATION

Arguments

- **row_index** (Integer)

The row index of the data to return. **Note that indices start at 0, not 1**.
0 <= row_index < Interpolation.total

- **inid** (integer)

Independent Node or Node Set id.

- **idof (optional)** (integer)

Independent Degrees-of-Freedom. The default value is 123456.

- **twghtx (optional)** (real)

Weighting factor for INID. Scales the x-translational component. The default value is 1.0.

- **twghty (optional)** (real)

Weighting factor for INID. Scales the y-translational component. The default value is twghtx.

- **twghtz (optional)** (real)

Weighting factor for INID. Scales the z-translational component. The default value is twghtx.

- **rwghtx (optional)** (real)

Weighting factor for INID. Scales the x-rotational component. The default value is twghtx.

- **rwghty (optional)** (real)

Weighting factor for INID. Scales the y-rotational component. The default value is twghtx.

- **rwghtz (optional)** (real)

Weighting factor for INID. Scales the z-rotational component. The default value is twghtx.

- **cidi (optional)** (integer)

Coordinate System ID if LOCAL option is active. The default value is 0

Returns

No return value

Example

To reset the values of row 3 of the keyword with INID 11, idof 1234, twghtx 2.2, twghty 4.2:

```
c_i.SetRowData(2,11,1234,2.2,4.2);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the constrained interpolation. The constrained interpolation will be sketched until you either call Interpolation.Unsketch(), Interpolation.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the constrained interpolation is sketched. If omitted redraw is true. If you want to sketch several constrained interpolations and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch constrained interpolation c_i:

```
c_i.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged constrained interpolations in the model. The constrained interpolations will be sketched until you either call Interpolation.Unsketch(), Interpolation.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged constrained interpolations will be sketched in

- **flag** (Flag)

Flag set on the constrained interpolations that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the constrained interpolations are sketched. If omitted redraw is true. If you want to sketch flagged constrained interpolations several times and only redraw after the last one then use false for

redraw and call View.Redraw().

Returns

No return value

Example

To sketch all constrained interpolations flagged with flag in model m:

```
Interpolation.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of constrained interpolations in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing constrained interpolations should be counted. If false or omitted referenced but undefined constrained interpolations will also be included in the total.

Returns

number of constrained interpolations

Return type

Number

Example

To get the total number of constrained interpolations in model m:

```
var total = Interpolation.Total(m);
```

Unblank()

Description

Unblanks the constrained interpolation

Arguments

No arguments

Returns

No return value

Example

To unblank constrained interpolation c_i:

```
c_i.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the constrained interpolations in the model.

Arguments

- **Model** (Model)

Model that all constrained interpolations will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the constrained interpolations in model m:

```
Interpolation.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged constrained interpolations in the model.

Arguments

- **Model** (Model)

Model that the flagged constrained interpolations will be unblanked in

- **flag** (Flag)

Flag set on the constrained interpolations that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the constrained interpolations in model m flagged with f:

```
Interpolation.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the constrained interpolations in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all constrained interpolations will be unset in

- **flag** (Flag)

Flag to unset on the constrained interpolations

Returns

No return value

Example

To unset the flag f on all the constrained interpolations in model m:

```
Interpolation.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the constrained interpolation.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the constrained interpolation is unsketched. If omitted redraw is true. If you want to unsketch several constrained interpolations and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch constrained interpolation c_i:

```
c_i.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all constrained interpolations.

Arguments

- **Model** (Model)

Model that all constrained interpolations will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the constrained interpolations are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all constrained interpolations in model m:

```
Interpolation.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged constrained interpolations in the model.

Arguments

- **Model** (Model)

Model that all constrained interpolations will be unsketched in

- **flag** (Flag)

Flag set on the constrained interpolations that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the constrained interpolations are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all constrained interpolations flagged with flag in model m:

```
Interpolation.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

Interpolation object.

Return type

Interpolation

Example

To check if Interpolation property c_i.example is a parameter by using the Interpolation.GetParameter() method:

```
if (c_i.ViewParameters().GetParameter(c_i.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for constrained interpolation with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The constrained interpolation object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for constrained interpolation c_i:

```
c_i.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for constrained interpolation c_i:

```
c_i.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for constrained interpolation. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for constrained interpolation c_i:

```
c_i.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this constrained interpolation.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for constrained interpolation c_i:

```
var xrefs = c_i.Xrefs();
```

---

toString()

Description

Creates a string containing the Interpolation data in keyword format. Note that this contains the keyword header and the keyword cards. See also Interpolation.Keyword() and Interpolation.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for Interpolation c_i in keyword format

```
var s = c_i.toString();
```

InterpolationSpotweld (Spr3) class

The InterpolationSpotweld class gives you access to constrained Interpolation Spotweld (spr3) cards in PRIMER.
More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- First(Model*[Model]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

InterpolationSpotweld constants

Constants for Flags for Interpolation

Name	Description
InterpolationSpotweld.INVERSE	Property INTP value EQ.2.0: Inverse distance weighting.
InterpolationSpotweld.LINEAR	Property INTP value EQ.0.0: Linear (default).
InterpolationSpotweld.UNIFORM	Property INTP value EQ.1.0: Uniform.

Constants for Material behaviour and damage model

Name	Description
InterpolationSpotweld.SPR3	SPR3 (default)
InterpolationSpotweld.SPR3_MAT_PARAM	SPR3 with selected material parameters as functions
InterpolationSpotweld.SPR3_MAT_PARAM_MOD	SPR3 with selected material parameters as functions and slight modification
InterpolationSpotweld.SPR4	SPR4
InterpolationSpotweld.SPR4_MAT_PARAM	SPR4 with selected material parameters as functions
InterpolationSpotweld.SPR4_MAT_PARAM_MOD	SPR4 with selected material parameters as functions and slight modification

InterpolationSpotweld properties

Name	Type	Description
alpha1	real/integer	Scaling factor alpha 1. Function ID if MODEL > 10.
alpha2	real	Plastic initiation displacement scaling factor alpha2.
alpha3	real	Plastic initiation displacement scaling factor alpha3.
bdmodel	real	Material behaviour and damage model. Values can be InterpolationSpotweld.SPR3, InterpolationSpotweld.SPR4, InterpolationSpotweld.SPR3_MAT_PARAM, InterpolationSpotweld.SPR4_MAT_PARAM, InterpolationSpotweld.SPR3_MAT_PARAM_MOD or InterpolationSpotweld.SPR4_MAT_PARAM_MOD
beta	real	Exponent for plastic potential beta 1. Function ID if bdmodel > 10.
beta2	real	Exponent for plastic initiation displacement beta2.
beta3	real	Exponent for plastic initiation displacement beta3.
damp	real	Viscous coefficient to model stiffness-proportional damping effects
defc	real	Fading energy for damage. Function ID if bdmodel > 30.
dens	real	Spotweld density (necessary for time step calculation).
exfc	real	Failure function exponent. Function ID if bdmodel > 30.
exists (read only)	logical	true if interpolation spotweld exists, false if referred to but not defined.
ffb	real	Resultant bending force at failure. Function ID if bdmodel > 30.
ffn	real	Resultant normal force at failure. Function ID if bdmodel > 30.
ffs	real	Resultant shear force at failure. Function ID if bdmodel > 30.
gamma	real	Scaling factor. Function ID if bdmodel > 30.
heading	string	Title
include	integer	The Include file number that the interpolation spotweld is in.

intp	real	Flag for interpolation. Values can be InterpolationSpotweld.LINEAR, InterpolationSpotweld.UNIFORM or InterpolationSpotweld.INVERSE.
lcdexp	integer	Load curve ID for damage exponent vs. mode mixity
lcf	integer	Load curve ID describing force versus plastic displacement.
lcupf	integer	Load curve ID describing plastic initiation displacement versus mode mixity. Required only for material behaviour and damage models InterpolationSpotweld.SPR3, InterpolationSpotweld.SPR3_MAT_PARAM or InterpolationSpotweld.SPR3_MAT_PARAM_MOD.
lcupr	integer	Load curve ID describing plastic rupture displacement versus mode mixity. Required only for material behaviour and damage models InterpolationSpotweld.SPR3, InterpolationSpotweld.SPR3_MAT_PARAM or InterpolationSpotweld.SPR3_MAT_PARAM_MOD.
mfsfc	real	Scaling factor for torsion term in resultant shear force. Function ID if bdmodel > 30.
model (read only)	integer	The Model number that the interpolation spotweld is in.
mrn	real	Proportionality factor for dependency RN.
mrs	real	Proportionality factor for dependency RS.
npfc	real	Plastic displacement offset for damage initiation. Function ID if bdmodel > 30.
nsid	integer	Node Set ID of spotweld location nodes.
pid1	integer	Part ID of first sheet.
pid2	integer	Part ID
pidvb	real	Part ID for visualization beams representing SPR3 in post-processing.
r	real	Spotweld Radius.
rn	real/integer	Tensile strength factor or negative Load curve with ID giving as a function of peel ratio . Function ID if MODEL > 10.
rs	real	Shear strength factor. Function ID if bdmodel > 10.
scarn	real	Scale factor for tensile strength factor RN
scars	real	Scale factor for tensile strength factor RS
sproff	real	Specifies the relative location of the spot weld between PID1 and PID2
sropt	real	Shear rotation option.
stiff	real/integer	Elastic stiffness OR material ID if less than 0. Function ID if MODEL > 10.
stiff2	real	Elastic shear stiffness. Function ID if bdmodel > 30.
stiff3	real	Elastic bending stiffness. Function ID if bdmodel > 30.
stiff4	real	Elastic torsional stiffness. Function ID if bdmodel > 30.
stifp	real	Plastic stiffness. Function ID if bdmodel > 30.
thick	real	Total thickness of both sheets.
upfn	real	Plastic initiation displacement in normal direction.
upfs	real	Plastic initiation displacement in shear direction.
uprn	real	Plastic rupture displacement in normal direction.
uprs	real	Plastic rupture displacement in shear direction.

Detailed Description

The InterpolationSpotweld class allows you to create, modify, edit and manipulate constrained interpolation spotweld (spr3) cards. See the documentation below for more details.

For convenience "Spr3" can also be used as the class name instead of "InterpolationSpotweld".

Constructor

new InterpolationSpotweld(Model*[Model]*, pid1*[integer]*, pid2*[integer]*, nsid*[integer]*, heading (optional)*[string]*)

Description

Create a new InterpolationSpotweld object.

Arguments

- **Model** (Model)

Model that constrained interpolation spotweld will be created in

- **pid1** (integer)

Part ID of first sheet.

- **pid2** (integer)

Part ID of second sheet.

- **nsid** (integer)

Node Set ID of spotweld location nodes.

- **heading (optional)** (string)

Title for the constrained interpolation spotweld

Returns

InterpolationSpotweld object

Return type

InterpolationSpotweld

Example

To create a new constrained interpolation spotweld in model m with first sheet 100, second sheet 200 and spotweld node set 100

```
var s = new InterpolationSpotweld(m, 100, 200, 100);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a interpolation spotweld.

Arguments

- **Comment** (Comment)

Comment that will be attached to the interpolation spotweld

Returns

No return value

Example

To associate comment c to the interpolation spotweld s:

```
s.AssociateComment(c);
```

Blank()

Description

Blanks the interpolation spotweld

Arguments

No arguments

Returns

No return value

Example

To blank interpolation spotweld s:

```
s.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the interpolation spotwelds in the model.

Arguments

- **Model** (Model)

Model that all interpolation spotwelds will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the interpolation spotwelds in model m:

```
InterpolationSpotweld.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged interpolation spotwelds in the model.

Arguments

- **Model** (Model)

Model that all the flagged interpolation spotwelds will be blanked in

- **flag** (Flag)

Flag set on the interpolation spotwelds that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the interpolation spotwelds in model m flagged with f:

```
InterpolationSpotweld.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the interpolation spotweld is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if interpolation spotweld s is blanked:

```
if (s.Blanked() ) do_something...
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the interpolation spotweld.

Arguments

- **flag** (Flag)

Flag to clear on the interpolation spotweld

Returns

No return value

Example

To clear flag f for interpolation spotweld s:

```
s.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the interpolation spotweld. The target include of the copied interpolation spotweld can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

InterpolationSpotweld object

Return type

InterpolationSpotweld

Example

To copy interpolation spotweld s into interpolation spotweld z:

```
var z = s.Copy();
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a interpolation spotweld.

Arguments

- **Comment** (Comment)

Comment that will be detached from the interpolation spotweld

Returns

No return value

Example

To detach comment c from the interpolation spotweld s:

```
s.DetachComment(c);
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for interpolation spotweld with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The interpolation spotweld object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for interpolation spotweld s:

```
s.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for interpolation spotweld s:

```
s.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for interpolation spotweld. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for interpolation spotweld s:

```
s.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first interpolation spotweld in the model.

Arguments

- **Model** (Model)

Model to get first interpolation spotweld in

Returns

InterpolationSpotweld object (or null if there are no interpolation spotwelds in the model).

Return type

InterpolationSpotweld

Example

To get the first interpolation spotweld in model m:

```
var s = InterpolationSpotweld.First(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the interpolation spotwelds in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all interpolation spotwelds will be flagged in

- **flag** (Flag)

Flag to set on the interpolation spotwelds

Returns

No return value

Example

To flag all of the interpolation spotwelds with flag f in model m:

```
InterpolationSpotweld.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the interpolation spotweld is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the interpolation spotweld

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if interpolation spotweld s has flag f set on it:

```
if (s.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each interpolation spotweld in the model.
Note that ForEach has been designed to make looping over interpolation spotwelds as fast as possible and so has some limitations.
Firstly, a single temporary InterpolationSpotweld object is created and on each function call it is updated with the current interpolation spotweld data. This means that you should not try to store the InterpolationSpotweld object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new interpolation spotwelds inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all interpolation spotwelds are in

- **func** (function)

Function to call for each interpolation spotweld

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the interpolation spotwelds in model m:

```
InterpolationSpotweld.ForEach(m, test);
function test(s)
{
// s is InterpolationSpotweld object
}
```

To call function test for all of the interpolation spotwelds in model m with optional object:

```
var data = { x:0, y:0 };
InterpolationSpotweld.ForEach(m, test, data);
function test(s, extra)
{
// s is InterpolationSpotweld object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of InterpolationSpotweld objects or properties for all of the interpolation spotwelds in a model in PRIMER. If the optional property argument is not given then an array of InterpolationSpotweld objects is returned. If the property argument is given, that property value for each interpolation spotweld is returned in the array instead of a InterpolationSpotweld object

Arguments

- **Model** (Model)

Model to get interpolation spotwelds from

- **property (optional)** (string)

Name for property to get for all interpolation spotwelds in the model

Returns

Array of InterpolationSpotweld objects or properties

Return type

Array

Example

To make an array of InterpolationSpotweld objects for all of the interpolation spotwelds in model m:

```
var a = InterpolationSpotweld.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each interpolation spotweld in model m:

```
var a = InterpolationSpotweld.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a interpolation spotweld.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the interpolation spotweld s:

```
var comm_array = s.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of InterpolationSpotweld objects for all of the flagged interpolation spotwelds in a model in PRIMER If the optional property argument is not given then an array of InterpolationSpotweld objects is returned. If the property argument is given, then that property value for each interpolation spotweld is returned in the array instead of a InterpolationSpotweld object

Arguments

- **Model** (Model)

Model to get interpolation spotwelds from

- **flag** (Flag)

Flag set on the interpolation spotwelds that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged interpolation spotwelds in the model

Returns

Array of InterpolationSpotweld objects or properties

Return type

Array

Example

To make an array of InterpolationSpotweld objects for all of the interpolation spotwelds in model m flagged with f:

```
var s = InterpolationSpotweld.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the interpolation spotwelds in model m flagged with f:

```
var a = InterpolationSpotweld.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the InterpolationSpotweld object for a interpolation spotweld ID.

Arguments

- **Model** (Model)

Model to find the interpolation spotweld in

- **number** (integer)

number of the interpolation spotweld you want the InterpolationSpotweld object for

Returns

InterpolationSpotweld object (or null if interpolation spotweld does not exist).

Return type

InterpolationSpotweld

Example

To get the InterpolationSpotweld object for interpolation spotweld 100 in model m

```
var s = InterpolationSpotweld.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a InterpolationSpotweld property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the InterpolationSpotweld.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

interpolation spotweld property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if InterpolationSpotweld property s.example is a parameter:

```
Options.property_parameter_names = true;
if (s.GetParameter(s.example) ) do_something...
Options.property_parameter_names = false;
```

To check if InterpolationSpotweld property s.example is a parameter by using the GetParameter method:

```
if (s.ViewParameters().GetParameter(s.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this interpolation spotweld (*CONSTRAINED_INTERPOLATION_SPOTWELD). **Note that a carriage return is not added**. See also InterpolationSpotweld.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for interpolation spotweld s:

```
var key = s.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the interpolation spotweld. **Note that a carriage return is not added**. See also InterpolationSpotweld.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for interpolation spotweld s:

```
var cards = s.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last interpolation spotweld in the model.

Arguments

- **Model** (Model)

Model to get last interpolation spotweld in

Returns

InterpolationSpotweld object (or null if there are no interpolation spotwelds in the model).

Return type

InterpolationSpotweld

Example

To get the last interpolation spotweld in model m:

```
var s = InterpolationSpotweld.Last(m);
```

Next()

Description

Returns the next interpolation spotweld in the model.

Arguments

No arguments

Returns

InterpolationSpotweld object (or null if there are no more interpolation spotwelds in the model).

Return type

InterpolationSpotweld

Example

To get the interpolation spotweld in model m after interpolation spotweld s:

```
var s = s.Next();
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a interpolation spotweld.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only interpolation spotwelds from that model can be picked. If the argument is a Flag then only interpolation spotwelds that are flagged with *limit* can be selected. If omitted, or null, any interpolation spotwelds from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

InterpolationSpotweld object (or null if not picked)

Return type

InterpolationSpotweld

Example

To pick a interpolation spotweld from model m giving the prompt 'Pick interpolation spotweld from screen':

```
var s = InterpolationSpotweld.Pick('Pick interpolation spotweld from screen',
m);
```

Previous()

Description

Returns the previous interpolation spotweld in the model.

Arguments

No arguments

Returns

InterpolationSpotweld object (or null if there are no more interpolation spotwelds in the model).

Return type

InterpolationSpotweld

Example

To get the interpolation spotweld in model m before interpolation spotweld s:

```
var s = s.Previous();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select interpolation spotwelds using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting interpolation spotwelds

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only interpolation spotwelds from that model can be selected. If the argument is a Flag then only interpolation spotwelds that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any interpolation spotwelds can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of interpolation spotwelds selected or null if menu cancelled

Return type

Number

Example

To select interpolation spotwelds from model m, flagging those selected with flag f, giving the prompt 'Select interpolation spotwelds':

```
InterpolationSpotweld.Select(f, 'Select interpolation spotwelds', m);
```

To select interpolation spotwelds, flagging those selected with flag f but limiting selection to interpolation spotwelds flagged with flag l, giving the prompt 'Select interpolation spotwelds':

```
InterpolationSpotweld.Select(f, 'Select interpolation spotwelds', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the interpolation spotweld.

Arguments

- **flag** (Flag)

Flag to set on the interpolation spotweld

Returns

No return value

Example

To set flag f for interpolation spotweld s:

```
s.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the interpolation spotweld. The interpolation spotweld will be sketched until you either call InterpolationSpotweld.Unsketch(), InterpolationSpotweld.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the interpolation spotweld is sketched. If omitted redraw is true. If you want to sketch several interpolation spotwelds and only redraw after the last one then use false for redraw and call

View.Redraw().

Returns

No return value

Example

To sketch interpolation spotweld s:

```
s.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged interpolation spotwelds in the model. The interpolation spotwelds will be sketched until you either call InterpolationSpotweld.Unsketch(), InterpolationSpotweld.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged interpolation spotwelds will be sketched in

- **flag** (Flag)

Flag set on the interpolation spotwelds that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the interpolation spotwelds are sketched. If omitted redraw is true. If you want to sketch flagged interpolation spotwelds several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all interpolation spotwelds flagged with flag in model m:

```
InterpolationSpotweld.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of interpolation spotwelds in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing interpolation spotwelds should be counted. If false or omitted referenced but undefined interpolation spotwelds will also be included in the total.

Returns

number of interpolation spotwelds

Return type

Number

Example

To get the total number of interpolation spotwelds in model m:

```
var total = InterpolationSpotweld.Total(m);
```

Unblank()

Description

Unblanks the interpolation spotweld

Arguments

No arguments

Returns

No return value

Example

To unblank interpolation spotweld s:

```
s.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the interpolation spotwelds in the model.

Arguments

- **Model** (Model)

Model that all interpolation spotwelds will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the interpolation spotwelds in model m:

```
InterpolationSpotweld.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged interpolation spotwelds in the model.

Arguments

- **Model** (Model)

Model that the flagged interpolation spotwelds will be unblanked in

- **flag** (Flag)

Flag set on the interpolation spotwelds that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the interpolation spotwelds in model m flagged with f:

```
InterpolationSpotweld.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the interpolation spotwelds in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all interpolation spotwelds will be unset in

- **flag** (Flag)

Flag to unset on the interpolation spotwelds

Returns

No return value

Example

To unset the flag f on all the interpolation spotwelds in model m:

```
InterpolationSpotweld.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the interpolation spotweld.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the interpolation spotweld is unsketched. If omitted redraw is true. If you want to unsketch several interpolation spotwelds and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch interpolation spotweld s:

```
s.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all interpolation spotwelds.

Arguments

- **Model** (Model)

Model that all interpolation spotwelds will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the interpolation spotwelds are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all interpolation spotwelds in model m:

```
InterpolationSpotweld.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged interpolation spotwelds in the model.

Arguments

- **Model** (Model)

Model that all interpolation spotwelds will be unsketched in

- **flag** (Flag)

Flag set on the interpolation spotwelds that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the interpolation spotwelds are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all interpolation spotwelds flagged with flag in model m:

```
InterpolationSpotweld.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

InterpolationSpotweld object.

Return type

InterpolationSpotweld

Example

To check if InterpolationSpotweld property s.example is a parameter by using the InterpolationSpotweld.GetParameter() method:

```
if (s.ViewParameters().GetParameter(s.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for interpolation spotweld with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The interpolation spotweld object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for interpolation spotweld s:

```
s.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for interpolation spotweld s:

```
s.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for interpolation spotweld. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for interpolation spotweld s:

```
s.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this interpolation spotweld.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for interpolation spotweld s:

```
var xrefs = s.Xrefs();
```

---

toString()

Description

Creates a string containing the interpolation spotweld data in keyword format. Note that this contains the keyword header and the keyword cards. See also InterpolationSpotweld.Keyword() and InterpolationSpotweld.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for interpolation spotweld s in keyword format

```
var str = s.toString();
```

Joint class

The Joint class gives you access to constrained joint cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- ExtractColour()
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)

- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

Joint constants

Name	Description
Joint.CONSTANT_VELOCITY	CONSTANT_VELOCITY is *CONSTRAINED_JOINT_CONSTANT_VELOCITY.
Joint.CYLINDRICAL	CYLINDRICAL is *CONSTRAINED_JOINT_CYLINDRICAL.
Joint.GEARS	GEARS is *CONSTRAINED_JOINT_GEARS.
Joint.LOCKING	LOCKING is *CONSTRAINED_JOINT_LOCKING.
Joint.PLANAR	PLANAR is *CONSTRAINED_JOINT_PLANAR.
Joint.PULLEY	PULLEY is *CONSTRAINED_JOINT_PULLEY.
Joint.RACK_AND_PINION	RACK_AND_PINION is *CONSTRAINED_JOINT_RACK_AND_PINION.
Joint.REVOLUTE	REVOLUTE is *CONSTRAINED_JOINT_REVOLUTE.
Joint.ROTATIONAL_MOTOR	ROTATIONAL_MOTOR is *CONSTRAINED_JOINT_ROTATIONAL_MOTOR.
Joint.SCREW	SCREW is *CONSTRAINED_JOINT_SCREW.
Joint.SPHERICAL	SPHERICAL is *CONSTRAINED_JOINT_SPHERICAL.
Joint.TRANSLATIONAL	TRANSLATIONAL is *CONSTRAINED_JOINT_TRANSLATIONAL.
Joint.TRANSLATIONAL_MOTOR	TRANSLATIONAL_MOTOR is *CONSTRAINED_JOINT_TRANSLATIONAL_MOTOR.
Joint.UNIVERSAL	UNIVERSAL is *CONSTRAINED_JOINT_UNIVERSAL.

Joint properties

Name	Type	Description
cid	integer	Coordinate system number.
colour	Colour	The colour of the joint
coupl	real	Coupling between force and moment failure.
damp	real	Damping scale factor.
exists (read only)	logical	true if joint exists, false if referred to but not defined.
failure	logical	true if _FAILURE option is set, false if not.
h_angle	real	Helix angle for gears.
heading	string	Constrained joint heading.
id	logical	true if _ID option is set, false if not
include	integer	The Include file number that the joint is in.
jid	integer	Constrained joint number (identical to label).
label	integer	Constrained joint number.
lcid	integer	Loadcuve number.

local	logical	true if _LOCAL option is set, false if not.
lst	integer	Local system type is accelerometer if lst is 1, rigid body if 0.
model (read only)	integer	The Model number that the joint is in.
mxx	real	Torsional moment resultant at failure.
myy	real	Moment resultant at failure.
mzz	real	Moment resultant at failure.
n1	integer	Node number 1.
n2	integer	Node number 2.
n3	integer	Node number 3.
n4	integer	Node number 4.
n5	integer	Node number 5.
n6	integer	Node number 6.
nxx	real	Axial force resultant at failure.
nyy	real	Force resultant at failure.
nzz	real	Force resultant at failure.
option	constant	The Constrained Joint option. Can be: Joint.SPHERICAL, Joint.REVOLUTE, Joint.CYLINDRICAL, Joint.PLANAR, Joint.UNIVERSAL, Joint.TRANSLATIONAL, Joint.LOCKING, Joint.TRANSLATIONAL_MOTOR, Joint.ROTATIONAL_MOTOR, Joint.GEARS, Joint.RACK_AND_PINION, Joint.CONSTANT_VELOCITY, Joint.PULLEY or Joint.SCREW
parm	real	Parameter for function.
r1	real	Gear and pulley radius.
raid	integer	Rigid body or accelerometer number.
rps	real	Relative penalty stiffness.
tfail	real	Time for joint failure.
type	integer	Flag for motor type.

Detailed Description

The Joint class allows you to create, modify, edit and manipulate constrained joint cards. See the documentation below for more details.

Constructor

new Joint(Model*[Model]*, options *[object]*)

Description

Create a new Joint object

Arguments

- **Model** (Model)

Model that constrained joint will be created in

- **options** (object)

Options for creating the joint

Object has the following properties:

Name	Type	Description
heading (optional)	string	Constrained joint title
id (optional)	integer	Constrained joint ID. If omitted, the joint will be created without the _ID option
nodes	Array of numbers	Array of Node IDs for the joint. At least 2 nodes must be given
type	constant	Constrained joint type (any)

Returns

Joint object

Return type

Joint

Example

To create a new constrained joint 500 called "test spherical joint" of type _SPHERICAL in model m with nodes 50 and 150

```
var j = new Joint(m, { type: Joint.SPHERICAL, nodes: [50, 150], id: 500,
heading: "test spherical joint" } );
```

To create a new constrained joint 500 called "test revolute joint" of type _REVOLUTE in model m with nodes 50, 100, 150 and 200

```
var j = new Joint(m, { type: Joint.REVOLUTE, nodes: [50, 100, 150, 200], id:
500, heading: "test revolute joint" } );
```

To create a new constrained joint 500 called "test translational joint" of type _TRANSLATIONAL in model m with nodes 50, 100, 150, 200, 250 and 300

```
var j = new Joint(m, { type: Joint.TRANSLATIONAL, nodes: [50, 100, 150, 200,
250, 300], id: 500, heading: "test translational joint" } );
```

new Joint(Model*[Model]*, option*[constant]*, n1*[integer]*, n2*[integer]*, jid (optional)*[integer]*, heading (optional)*[string]*) [deprecated]

This function is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Create a new Joint object.

Arguments

- **Model** (Model)

Model that constrained joint will be created in

- **option** (constant)

Constrained joint type (any).

- **n1** (integer)

Node 1.

- **n2** (integer)

Node 2.

- **jid (optional)** (integer)

Constrained joint number.

- **heading (optional)** (string)

Constrained joint title.

Returns

Joint object

Return type

Joint

Example

To create a new constrained joint 500 called "test spherical joint" of type _SPHERICAL in model m with nodes 50 and 150

```
var j = new Joint(m, Joint.SPHERICAL, 50, 150, 500, "test spherical joint");
```

To create a new constrained joint 500 called "test revolute joint" of type _REVOLUTE in model m with nodes 50, 100, 150 and 200

```
var j = new Joint(m, Joint.REVOLUTE, 50, 100, 500, "test revolute joint");
j.n3 = 150;
j.n4 = 200;
```

new Joint(Model*[Model]*, option*[constant]*, n1*[integer]*, n2*[integer]*, n3*[integer]*, n4*[integer]*, jid (optional)*[integer]*, heading (optional)*[string]*) [deprecated]

This function is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Create a new Joint object.

Arguments

- **Model** (Model)

Model that constrained joint will be created in

- **option** (constant)

Constrained joint type. Can be Joint.REVOLUTE, Joint.CYLINDRICAL, Joint.PLANAR, Joint.UNIVERSAL or Joint.TRANSLATIONAL_MOTOR.

- **n1** (integer)

Node 1.

- **n2** (integer)

Node 2.

- **n3** (integer)

Node 3.

- **n4** (integer)

Node 4.

- **jid (optional)** (integer)

Constrained joint number.

- **heading (optional)** (string)

Constrained joint title.

Returns

Joint object

Return type

Joint

Example

To create a new constrained joint 500 called "test revolute joint" of type _REVOLUTE in model m with nodes 50, 100, 150 and 200

```
var j = new Joint(m, Joint.REVOLUTE, 50, 100, 150, 200, 500, "test revolute joint");
```

new Joint(Model*[Model]*, option*[constant]*, n1*[integer]*, n2*[integer]*, n3*[integer]*, n4*[integer]*, n5*[integer]*, n6*[integer]*, jid (optional)*[integer]*, heading (optional)*[string]*) [deprecated]

This function is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Create a new Joint object.

Arguments

- **Model** (Model)

Model that constrained joint will be created in

- **option** (constant)

Constrained joint type. Can be Joint.TRANSLATIONAL, Joint.LOCKING, Joint.ROTATIONAL_MOTOR, Joint.GEARS, Joint.RACK_AND_PINION, Joint.CONSTANT_VELOCITY, Joint.PULLEY or Joint.SCREW.

- **n1** (integer)

Node 1.

- **n2** (integer)

Node 2.

- **n3** (integer)

Node 3.

- **n4** (integer)

Node 4.

- **n5** (integer)

Node 5.

- **n6** (integer)

Node 6.

- **jid (optional)** (integer)

Constrained joint number.

- **heading (optional)** (string)

Constrained joint title.

Returns

Joint object

Return type

Joint

Example

To create a new constrained joint 500 called "test translational joint" of type _TRANSLATIONAL in model m with nodes 50, 100, 150, 300, 250 and 300

```
var j = new Joint(m, Joint.TRANSLATIONAL, 50, 100, 150, 200, 250, 300, 500,
"test translational joint");
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a joint.

Arguments

- **Comment** (Comment)

Comment that will be attached to the joint

Returns

No return value

Example

To associate comment c to the joint j:

```
j.AssociateComment(c);
```

Blank()

Description

Blanks the joint

Arguments

No arguments

Returns

No return value

Example

To blank joint j:

```
j.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the joints in the model.

Arguments

- **Model** (Model)

Model that all joints will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the joints in model m:

```
Joint.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged joints in the model.

Arguments

- **Model** (Model)

Model that all the flagged joints will be blanked in

- **flag** (Flag)

Flag set on the joints that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the joints in model m flagged with f:

```
Joint.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the joint is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if joint j is blanked:

```
if (j.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse joint j:

```
j.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the joint.

Arguments

- **flag** (Flag)

Flag to clear on the joint

Returns

No return value

Example

To clear flag f for joint j:

```
j.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the joint. The target include of the copied joint can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

Joint object

Return type

Joint

Example

To copy joint j into joint z:

```
var z = j.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a joint

Arguments

- **Model** (Model)

Model that the joint will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

Joint object (or null if not made).

Return type

Joint

Example

To start creating a joint in model m:

```
var j = Joint.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a joint.

Arguments

- **Comment** (Comment)

Comment that will be detached from the joint

Returns

No return value

Example

To detach comment c from the joint j:

```
j.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit joint j:

```
j.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for joint with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The joint object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for joint j:

```
j.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for joint j:

```
j.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for joint. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for joint j:

```
j.Error("My custom error");
```

ExtractColour()

Description

Extracts the **actual** colour used for joint.
By default in PRIMER many entities such as elements get their colour automatically from the part that they are in. PRIMER cycles through 13 default colours based on the label of the entity. In this case the joint colour property will return the value Colour.PART instead of the actual colour. This method will return the actual colour which is used for drawing the joint.

Arguments

No arguments

Returns

colour value (integer)

Return type

Number

Example

To return the colour used for drawing joint j:

```
var colour = j.ExtractColour();
```

First(Model*[Model]*) [static]

Description

Returns the first joint in the model.

Arguments

- **Model** (Model)

Model to get first joint in

Returns

Joint object (or null if there are no joints in the model).

Return type

Joint

Example

To get the first joint in model m:

```
var j = Joint.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free joint label in the model. Also see Joint.LastFreeLabel(), Joint.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free joint label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

Joint label.

Return type

Number

Example

To get the first free joint label in model m:

```
var label = Joint.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the joints in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all joints will be flagged in

- **flag** (Flag)

Flag to set on the joints

Returns

No return value

Example

To flag all of the joints with flag f in model m:

```
Joint.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the joint is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the joint

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if joint j has flag f set on it:

```
if (j.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each joint in the model.
Note that ForEach has been designed to make looping over joints as fast as possible and so has some limitations. Firstly, a single temporary Joint object is created and on each function call it is updated with the current joint data. This means that you should not try to store the Joint object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new joints inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all joints are in

- **func** (function)

Function to call for each joint

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the joints in model m:

```
Joint.ForEach(m, test);
function test(j)
{
// j is Joint object
}
```

To call function test for all of the joints in model m with optional object:

```
var data = { x:0, y:0 };
Joint.ForEach(m, test, data);
function test(j, extra)
{
// j is Joint object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Joint objects or properties for all of the joints in a model in PRIMER. If the optional property argument is not given then an array of Joint objects is returned. If the property argument is given, that property value for each joint is returned in the array instead of a Joint object

Arguments

- **Model** (Model)

Model to get joints from

- **property (optional)** (string)

Name for property to get for all joints in the model

Returns

Array of Joint objects or properties

Return type

Array

Example

To make an array of Joint objects for all of the joints in model m:

```
var a = Joint.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each joint in model m:

```
var a = Joint.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a joint.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the joint j:

```
var comm_array = j.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of Joint objects for all of the flagged joints in a model in PRIMER If the optional property argument is not given then an array of Joint objects is returned. If the property argument is given, then that property value for each joint is returned in the array instead of a Joint object

Arguments

- **Model** (Model)

Model to get joints from

- **flag** (Flag)

Flag set on the joints that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged joints in the model

Returns

Array of Joint objects or properties

Return type

Array

Example

To make an array of Joint objects for all of the joints in model m flagged with f:

```
var j = Joint.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the joints in model m flagged with f:

```
var a = Joint.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the Joint object for a joint ID.

Arguments

- **Model** (Model)

Model to find the joint in

- **number** (integer)

number of the joint you want the Joint object for

Returns

Joint object (or null if joint does not exist).

Return type

Joint

Example

To get the Joint object for joint 100 in model m

```
var j = Joint.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a Joint property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the Joint.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

joint property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if Joint property j.example is a parameter:

```
Options.property_parameter_names = true;
if (j.GetParameter(j.example) ) do_something...
Options.property_parameter_names = false;
```

To check if Joint property j.example is a parameter by using the GetParameter method:

```
if (j.ViewParameters().GetParameter(j.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this joint (*CONSTRAINED_JOINT). **Note that a carriage return is not added**. See also Joint.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for joint j:

```
var key = j.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the joint. **Note that a carriage return is not added**. See also Joint.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for joint j:

```
var cards = j.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last joint in the model.

Arguments

- **Model** (Model)

Model to get last joint in

Returns

Joint object (or null if there are no joints in the model).

Return type

Joint

Example

To get the last joint in model m:

```
var j = Joint.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free joint label in the model. Also see Joint.FirstFreeLabel(), Joint.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free joint label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

Joint label.

Return type

Number

Example

To get the last free joint label in model m:

```
var label = Joint.LastFreeLabel(m);
```

Next()

Description

Returns the next joint in the model.

Arguments

No arguments

Returns

Joint object (or null if there are no more joints in the model).

Return type

Joint

Example

To get the joint in model m after joint j:

```
var j = j.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) joint label in the model. Also see Joint.FirstFreeLabel(), Joint.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free joint label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

Joint label.

Return type

Number

Example

To get the next free joint label in model m:

```
var label = Joint.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a joint.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only joints from that model can be picked. If the argument is a Flag then only joints that are flagged with *limit* can be selected. If omitted, or null, any joints from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

Joint object (or null if not picked)

Return type

Joint

Example

To pick a joint from model m giving the prompt 'Pick joint from screen':

```
var j = Joint.Pick('Pick joint from screen', m);
```

Previous()

Description

Returns the previous joint in the model.

Arguments

No arguments

Returns

Joint object (or null if there are no more joints in the model).

Return type

Joint

Example

To get the joint in model m before joint j:

```
var j = j.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the joints in the model.

Arguments

- **Model** (Model)

Model that all joints will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the joints in model m, from 1000000:

```
Joint.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged joints in the model.

Arguments

- **Model** (Model)

Model that all the flagged joints will be renumbered in

- **flag** (Flag)

Flag set on the joints that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the joints in model m flagged with f, from 1000000:

```
Joint.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select joints using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting joints

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only joints from that model can be selected. If the argument is a Flag then only joints that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any joints can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of joints selected or null if menu cancelled

Return type

Number

Example

To select joints from model m, flagging those selected with flag f, giving the prompt 'Select joints':

```
Joint.Select(f, 'Select joints', m);
```

To select joints, flagging those selected with flag f but limiting selection to joints flagged with flag l, giving the prompt 'Select joints':

```
Joint.Select(f, 'Select joints', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the joint.

Arguments

- **flag** (Flag)

Flag to set on the joint

Returns

No return value

Example

To set flag f for joint j:

```
j.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the joint. The joint will be sketched until you either call Joint.Unsketch(), Joint.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the joint is sketched. If omitted redraw is true. If you want to sketch several joints and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch joint j:

```
j.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged joints in the model. The joints will be sketched until you either call Joint.Unsketch(), Joint.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged joints will be sketched in

- **flag** (Flag)

Flag set on the joints that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the joints are sketched. If omitted redraw is true. If you want to sketch flagged joints several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all joints flagged with flag in model m:

```
Joint.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of joints in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing joints should be counted. If false or omitted referenced but undefined joints will also be included in the total.

Returns

number of joints

Return type

Number

Example

To get the total number of joints in model m:

```
var total = Joint.Total(m);
```

Unblank()

Description

Unblanks the joint

Arguments

No arguments

Returns

No return value

Example

To unblank joint j:

```
j.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the joints in the model.

Arguments

- **Model** (Model)

Model that all joints will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the joints in model m:

```
Joint.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged joints in the model.

Arguments

- **Model** (Model)

Model that the flagged joints will be unblanked in

- **flag** (Flag)

Flag set on the joints that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the joints in model m flagged with f:

```
Joint.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the joints in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all joints will be unset in

- **flag** (Flag)

Flag to unset on the joints

Returns

No return value

Example

To unset the flag f on all the joints in model m:

```
Joint.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the joint.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the joint is unsketched. If omitted redraw is true. If you want to unsketch several joints and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch joint j:

```
j.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all joints.

Arguments

- **Model** (Model)

Model that all joints will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the joints are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all joints in model m:

```
Joint.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged joints in the model.

Arguments

- **Model** (Model)

Model that all joints will be unsketched in

- **flag** (Flag)

Flag set on the joints that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the joints are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all joints flagged with flag in model m:

```
Joint.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

Joint object.

Return type

Joint

Example

To check if Joint property j.example is a parameter by using the Joint.GetParameter() method:

```
if (j.ViewParameters().GetParameter(j.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for joint with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The joint object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for joint j:

```
j.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for joint j:

```
j.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for joint. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for joint j:

```
j.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this joint.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for joint j:

```
var xrefs = j.Xrefs();
```

---

toString()

Description

Creates a string containing the joint data in keyword format. Note that this contains the keyword header and the keyword cards. See also Joint.Keyword() and Joint.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for joint j in keyword format

```
var s = j.toString();
```

JointStiffness (Jstf) class

The JointStiffness class gives you access to constrained joint stiffness cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]

- Xrefs()
- toString()

JointStiffness constants

Name	Description
JointStiffness.CYLINDRICAL	JointStiffness is *CONSTRAINED_JOINT_STIFFNESS_CYLINDRICAL.
JointStiffness.FLEXION_ TORSION	JointStiffness is *CONSTRAINED_JOINT_STIFFNESS_ FLEXION-TORSION.
JointStiffness.GENERALIZED	JointStiffness is *CONSTRAINED_JOINT_STIFFNESS_GENERALIZED.
JointStiffness.TRANSLATIONAL	JointStiffness is *CONSTRAINED_JOINT_STIFFNESS_TRANSLATIONAL.

JointStiffness properties

Name	Type	Description
cida	integer	Coordinate System ID #A.
cidb	integer	Coordinate System ID #B.
dlcidal	integer	LC: Alpha damping moment vs Rotl vel.
dlcidbt	integer	LC: Beta damping moment vs Rotl vel.
dlcidg	integer	LC: Gamma damping factor vs factor on Alpha damping moment.
dlcidp	integer	LC: P damping vs P rel velocity.
dlcidph	integer	LC: Phi damping moment vs rotation vel.
dlcidps	integer	LC: Psi damping moment vs rotation vel.
dlcidr	integer	LC: R damping vs R rel velocity.
dlcidt	integer	LC: Theta damping moment vs rotation vel.
dlcidx	integer	LC: X damping vs X rel velocity.
dlcidy	integer	LC: Y damping vs Y rel velocity.
dlcidz	integer	LC: Z damping vs Z rel velocity.
esal	real	Stiffness/angle in Alpha direction.
esbt	real	Stiffness/angle in Beta direction.
esph	real	Stiffness/angle in Phi direction.
esps	real	Stiffness/angle in Psi direction.
esr	real	Elastic stiffness for R stop and friction.
est	real	Stiffness/angle in Theta direction.
esx	real	Elastic stiffness for X stop and friction.
esy	real	Elastic stiffness for Y stop and friction.
esz	real	Elastic stiffness for Z stop and friction.
exists (read only)	logical	true if joint stiffness exists, false if referred to but not defined.
fd	real	Dynamic friction coefficient.
ffr	integer	LC: Lim R force, or yield force vs R translation.
ffx	integer	LC: Lim X force, or yield force vs X translation.

ffy	integer	LC: Lim Y force, or yield force vs Y translation.
ffz	integer	LC: Lim Z force, or yield force vs Z translation.
fmal	integer	LC: Alpha Frictional moment vs rotation.
fmbt	integer	LC: Beta Frictional moment vs rotation.
fmph	integer	LC: Psi frictional moment vs rotation.
fmps	integer	LC: Psi frictional moment vs rotation.
fmt	integer	LC: Theta frictional moment vs rotation.
fs	real	Static friction coefficient.
heading	string	This property is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. JointStiffness heading. Use the title property instead [deprecated]
include	integer	The Include file number that the joint stiffness is in.
jid	integer	Joint for restraint/table uses.
jsid	integer	ID of the JointStiffness. Also see the label property which is an alternative name for this
label	integer	Label of the JointStiffness
lcidal	integer	LC: Alpha moment vs Rotation.
lcidbt	integer	LC: Beta moment vs Rotation.
lcidg	integer	LC: Gamma angle vs factor on Alpha blending.
lcidph	integer	LC: Phi moment vs rotation.
lcidps	integer	LC: Psi moment vs rotation.
lcidr	integer	LC: R force vs R rel displ.
lcidt	integer	LC: Theta moment vs rotation.
lcidx	integer	LC: X force vs X rel displ.
lcidy	integer	LC: Y force vs Y rel displ.
lcidz	integer	LC: Z force vs Z rel displ.
model (read only)	integer	The Model number that the joint stiffness is in.
nsabt	real	Stop angle for -ve Beta rotation.
nsaph	real	Stop angle for -ve Phi rotation.
nsaps	real	Stop angle for -ve Psi rotation.
nsat	real	Stop angle for -ve Theta rotation.
nsdx	real	Limiting -ve X translation.
nsdy	real	Limiting -ve Y translation.
nsdz	real	Limiting -ve Z translation.
option	constant	JointStiffness type. Can be JointStiffness.GENERALIZED, JointStiffness.FLEXION_TORSION, JointStiffness.TRANSLATIONAL or JointStiffness.CYLINDRICAL
pida	integer	Part ID #A.
pidb	integer	Part ID #B.
psabt	real	Stop angle for +ve Beta rotation.
psaph	real	Stop angle for +ve Phi rotation.
psaps	real	Stop angle for +ve Psi rotation.

psat	real	Stop angle for +ve Theta rotation.
psdr	real	Limiting R translation.
psdx	real	Limiting +ve X translation.
psdy	real	Limiting +ve Y translation.
psdz	real	Limiting +ve Z translation.
rad1	real	Radius of pin.
rad2	real	Radius of hole.
rps	real	Relative penalty stiffness.
saal	real	Stop angle for Alpha rotation.
title	string	JointStiffness title

Detailed Description

The JointStiffness class allows you to create, modify, edit and manipulate joint stiffness cards. See the documentation below for more details.

For convenience "Jstf" can also be used as the class name instead of "JointStiffness".

Constructor

new JointStiffness(Model*[Model]*, options *[object]*)

Description

Create a new JointStiffness object. The fields on card 1 of the joint stiffness can be set in the constructor using the option argument. To set any other values use properties.

Arguments

- **Model** (Model)

Model that the joint stiffness will be created in

- **options** (object)

Options for creating the joint stiffness

Object has the following properties:

Name	Type	Description
cida (optional)	integer	Coordinate System ID #A
cidb (optional)	integer	Coordinate System ID #B
id	integer	Joint stiffness ID
jid (optional)	integer	Joint for restraint/table uses
option	constant	Constrained joint stiffness option. Can be JointStiffness.GENERALIZED, JointStiffness.FLEXION_TORSION, JointStiffness.TRANSLATIONAL or JointStiffness.CYLINDRICAL
pida (optional)	integer	Part ID #A
pidb (optional)	integer	Part ID #B
rps (optional)	real	Relative penalty stiffness (for JointStiffness.TRANSLATIONAL or JointStiffness.CYLINDRICAL)

title (optional)	string	Joint stiffness title

Returns

JointStiffness object

Return type

JointStiffness

Example

To create a new joint stiffness 500 with title "test" of type GENERALIZED in model m with rigid body parts 50 and 150 and lcidph 10

```
var j = new JointStiffness(m, { type: JointStiffness.GENERALIZED, id: 500, pida: 50, pidb: 50, title: "test" } );
j.lcidph = 10;
```

new JointStiffness(Model*[Model]*, option*[constant]*, label*[integer]*, pida*[integer]*, pidb*[integer]*, cida*[integer]*, cidb*[integer]*, jid*[integer]*, lcidph*[integer]*, lcidt*[integer]*, lcidps*[integer]*, dlcidph*[integer]*, dlcidt*[integer]*, dlcidps*[integer]*, esph*[real]*, fmph*[integer]*, est*[real]*, fmt*[integer]*, esps*[real]*, fmps*[integer]*, nsaph*[real]*, psaph*[real]*, nsat*[real]*, psat*[real]*, nsaps*[real]*, psaps*[real]*) [deprecated]

This function is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Create a new JointStiffness object for *CONSTRAINED_JOINT_STIFFNESS_GENERALIZED.

Arguments

- **Model** (Model)

Model that jstf will be created in

- **option** (constant)

Must be JointStiffness.GENERALIZED.

- **label** (integer)

JointStiffness ID of the JSTF. Also see the label argument which is an alternative name for this.

- **pida** (integer)

Part ID #A.

- **pidb** (integer)

Part ID #B.

- **cida** (integer)

Coordinate System ID #A.

- **cidb** (integer)

Coordinate System ID #B.

- **jid** (integer)

Joint for restraint/table uses.

- **lcidph** (integer)

LC: Phi moment vs rotation.

- **lcidt** (integer)

LC: Theta moment vs rotation.

- **lcidps** (integer)

LC: Psi moment vs rotation.

- **dlcidph** (integer)

LC: Phi damping moment vs rotation vel.

- **dlcidt** (integer)

LC: Theta damping moment vs rotation vel.

- **dlcidps** (integer)

LC: Psi damping moment vs rotation vel.

- **esph** (real)

Stiffness/angle in Phi direction.

- **fmph** (integer)

LC: Psi frictional moment vs rotation.

- **est** (real)

Stiffness/angle in Theta direction.

- **fmt** (integer)

LC: Theta frictional moment vs rotation.

- **esps** (real)

Stiffness/angle in Psi direction.

- **fmps** (integer)

LC: Psi frictional moment vs rotation.

- **nsaph** (real)

Stop angle for -ve Phi rotation.

- **psaph** (real)

Stop angle for +ve Phi rotation.

- **nsat** (real)

Stop angle for -ve Theta rotation.

- **psat** (real)

Stop angle for +ve Theta rotation.

- **nsaps** (real)

Stop angle for -ve Psi rotation.

- **psaps** (real)

Stop angle for +ve Psi rotation.

Returns

JointStiffness object

Return type

JointStiffness

Example

To create a new jstf 1000 of type GENERALIZED in model m with the following specifiction: pida, pidb, cida, cidb, jid are 91, 92, 81, 82, 71 respectively; lcidph, lcidt, lcidps, dlcidph, dlcidt, dlcidps are 1, 2, 3, 4, 5, 6 respectively; esph, fmph, est, fmt, esps, fmps are 11.0, 11, 12.0, 12, 13.0, 13 respectively; nsaph, psaph, nsat, psat, nsaps, psaps are -20, 20, -30, 30, -40, 40 respectively.

```
var j = new JointStiffness(m, JointStiffness.GENERALIZED, 1000, 91, 92, 81, 82,
71, 1, 2, 3, 4, 5, 6, 11.0, 11, 12.0, 12, 13.0, 13, -20, 20, -30, 30, -40, 40);
```

new JointStiffness(Model*[Model]*, option*[constant]*, label*[integer]*, pida*[integer]*, pidb*[integer]*, cida*[integer]*, cidb*[integer]*, jid*[integer]*, lcidal*[integer]*, lcidg*[integer]*, lcidbt*[integer]*, dlcidal*[integer]*, dlcidg*[integer]*, dlcidbt*[integer]*, esal*[real]*, fmal*[integer]*, esbt*[real]*, fmbt*[integer]*, saal*[real]*, nsabt*[real]*, psabt*[real]*) [deprecated]

This function is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Create a new JointStiffness object for *CONSTRAINED_JOINT_STIFFNESS_FLEXION-TORSION.

Arguments

- **Model** (Model)

Model that jstf will be created in

- **option** (constant)

Must be JointStiffness.FLEXION_TORSION.

- **label** (integer)

JointStiffness ID of the JSTF. Also see the label argument which is an alternative name for this.

- **pida** (integer)

Part ID #A.

- **pidb** (integer)

Part ID #B.

- **cida** (integer)

Coordinate System ID #A.

- **cidb** (integer)

Coordinate System ID #B.

- **jid** (integer)

Joint for restraint/table uses.

- **lcidal** (integer)

LC: Alpha moment vs Rotation.

- **lcidg** (integer)

LC: Gamma angle vs factor on Alpha blending.

- **lcidbt** (integer)

LC: Beta moment vs Rotation.

- **dlcidal** (integer)

LC: Alpha damping moment vs Rotl vel.

- **dlcidg** (integer)

LC: Gamma damping factor vs factor on Alpha damping moment.

- **dlcidbt** (integer)

LC: Beta damping moment vs Rotl vel.

- **esal** (real)

Stiffness/angle in Alpha direction.

- **fmal** (integer)

LC: Alpha Frictional moment vs rotation.

- **esbt** (real)

Stiffness/angle in Beta direction.

- **fmbt** (integer)

LC: Beta Frictional moment vs rotation.

- **saal** (real)

Stop angle for Alpha rotation.

- **nsabt** (real)

Stop angle for -ve Beta rotation.

- **psabt** (real)

Stop angle for +ve Beta rotation.

Returns

JointStiffness object

Return type

JointStiffness

Example

To create a new jstf 2000 of type GENERALIZED in model m with the following specifiction: pida, pidb, cida, cidb, jid are 81, 82, 71, 72, 61 respectively; lcidal, lcidg, lcidbt, dlcidal, dlcidg, dlcidbt are 1, 2, 3, 4, 5, 6 respectively; esal, fmal, esbt, fmbt are 11.5, 12, 12.5, 13 respectively; saal, nsabt, psabt are 22.5, 25.0, 27.5 respectively.

```
var j = new JointStiffness(m, JointStiffness.FLEXION_TORSION, 2000, 81, 82, 71,
72, 61, 1, 2, 3, 4, 5, 6, 11.5, 12, 12.5, 13, 22.5, 25.0, 27.5);
```

new JointStiffness(Model*[Model]*, option*[constant]*, label*[integer]*, pida*[integer]*, pidb*[integer]*, cida*[integer]*, cidb*[integer]*, jid*[integer]*, rps*[real]*, lcidx*[integer]*, lcidy*[integer]*, lcidz*[integer]*, dlcidx*[integer]*, dlcidy*[integer]*, dlcidz*[integer]*, esx*[real]*, ffx*[integer]*, esy*[real]*, ffy*[integer]*, esz*[real]*, ffz*[integer]*, nsdx*[real]*, psdx*[real]*, nsdy*[real]*, psdy*[real]*, nsdz*[real]*, psdz*[real]*, fs*[real]*, fd*[real]*) [deprecated]

This function is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Create a new JointStiffness object for *CONSTRAINED_JOINT_STIFFNESS_TRANSLATIONAL.

Arguments

- **Model** (Model)

Model that jstf will be created in

- **option** (constant)

Must be JointStiffness.TRANSLATIONAL.

- **label** (integer)

JointStiffness ID of the JSTF. Also see the label argument which is an alternative name for this.

- **pida** (integer)

Part ID #A.

- **pidb** (integer)

Part ID #B.

- **cida** (integer)

Coordinate System ID #A.

- **cidb** (integer)

Coordinate System ID #B.

- **jid** (integer)

Joint for restraint/table uses.

- **rps** (real)

Relative penalty stiffness.

- **lcidx** (integer)

LC: X force vs X rel displ.

- **lcidy** (integer)

LC: Y force vs Y rel displ.

- **lcidz** (integer)

LC: Z force vs Z rel displ.

- **dlcidx** (integer)

LC: X damping vs X rel velocity.

- **dlcidy** (integer)

LC: Y damping vs Y rel velocity.

- **dlcidz** (integer)

LC: Z damping vs Z rel velocity.

- **esx** (real)

Elastic stiffness for X stop and friction.

- **ffx** (integer)

LC: Lim X force, or yield force vs X translation.

- **esy** (real)

Elastic stiffness for Y stop and friction.

- **ffy** (integer)

LC: Lim Y force, or yield force vs Y translation.

- **esz** (real)

Elastic stiffness for Z stop and friction.

- **ffz** (integer)

LC: Lim Z force, or yield force vs Z translation.

- **nsdx** (real)

Limiting -ve X translation.

- **psdx** (real)

Limiting +ve X translation.

- **nsdy** (real)

Limiting -ve Y translation.

- **psdy** (real)

Limiting +ve Y translation.

- **nsdz** (real)

Limiting -ve Z translation.

- **psdz** (real)

Limiting +ve Z translation.

- **fs** (real)

Static friction coefficient.

- **fd** (real)

Dynamic friction coefficient.

Returns

JointStiffness object

Return type

JointStiffness

Example

To create a new jstf 3000 of type TRANSLATIONAL in model m with the following specifiction: pida, pidb, cida, cidb, jid, rps are 71, 72, 61, 62, 51, 2 respectively; lcidx, lcidy, lcidz, dlcidx, dlcidy, dlcidz are 1, 2, 3, 4, 5, 6 respectively; esx, ffx, esy, ffy, esz, ffz are are 12.5, 13, 13.5, 14, 14.5, 15 respectively; nsdx, psdx, nsdy, psdy, nsdz, psdz, fs, fd are -30, 30, -40, 40, -50, 50, 0.2, 0.1 respectively.

```
var j = new JointStiffness(m, JointStiffness.TRANSLATIONAL, 3000, 71, 72, 61,
62, 51, 2, 1, 2, 3, 4, 5, 6, 12.5, 13, 13.5, 14, 14.5, 15, -30, 30, -40, 40,
-50, 50, 0.2, 0.1);
```

new JointStiffness(Model*[Model]*, option*[constant]*, label*[integer]*, pida*[integer]*, pidb*[integer]*, cida*[integer]*, cidb*[integer]*, jid*[integer]*, rps*[real]*, lcidr*[integer]*, lcidz*[integer]*, dlcidr*[integer]*, dlcidp*[integer]*, dlcidz*[integer]*, lcidt*[integer]*, dlcidt*[integer]*, esr*[real]*, ffr*[integer]*, esz*[real]*, ffz*[integer]*, rad1*[real]*, rad2*[real]*, psdr*[real]*, nsdz*[real]*, psdz*[real]*, fs*[real]*, fd*[real]*) [deprecated]

This function is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Create a new JointStiffness object for *CONSTRAINED_JOINT_STIFFNESS_CYLINDRICAL.

Arguments

- **Model** (Model)

Model that jstf will be created in

- **option** (constant)

Must be JointStiffness.CYLINDRICAL.

- **label** (integer)

JointStiffness ID of the JSTF. Also see the label argument which is an alternative name for this.

- **pida** (integer)

Part ID #A.

- **pidb** (integer)

Part ID #B.

- **cida** (integer)

Coordinate System ID #A.

- **cidb** (integer)

Coordinate System ID #B.

- **jid** (integer)

Joint for restraint/table uses.

- **rps** (real)

Relative penalty stiffness.

- **lcidr** (integer)

LC: R force vs R rel displ.

- **lcidz** (integer)

LC: Z force vs Z rel displ.

- **dlcidr** (integer)

LC: R damping vs R rel velocity.

- **dlcidp** (integer)

LC: P damping vs P rel velocity.

- **dlcidz** (integer)

LC: Z damping vs Z rel velocity.

- **lcidt** (integer)

LC: Theta moment vs rotation.

- **dlcidt** (integer)

LC: Theta damping moment vs rotation vel.

- **esr** (real)

Elastic stiffness for R stop and friction.

- **ffr** (integer)

LC: Lim R force, or yield force vs R translation.

- **esz** (real)

Elastic stiffness for Z stop and friction.

- **ffz** (integer)

LC: Lim Z force, or yield force vs Z translation.

- **rad1** (real)

Radius of pin.

- **rad2** (real)

Radius of hole.

- **psdr** (real)

Limiting R translation.

- **nsdz** (real)

Limiting -ve Z translation.

- **psdz** (real)

Limiting +ve Z translation.

- **fs** (real)

Static friction coefficient.

- **fd** (real)

Dynamic friction coefficient.

Returns

JointStiffness object

Return type

JointStiffness

Example

To create a new jstf 4000 of type CYLINDRICAL in model m with the following specifiction: pida, pidb, cida, cidb, jid, rps are 61, 62, 51, 52, 41, 2 respectively; lcidr, lcidz, dlcidr, dlcidp, dlcidz, lcidt, dlcidt are 1, 2, 3, 4, 5, 6, 7 respectively; esr, ffr, esz, ffz, rad1, rad2 are are 12.5, 13, 13.5, 14, 14.5, 15.5 respectively; psdr, nsdz, psdz, fs, fd are 30, -40, 50, 0.2, 0.1 respectively.

```
var j = new JointStiffness(m, JointStiffness.CYLINDRICAL, 4000, 61, 62, 51, 52, 41, 2, 1, 2, 3, 4, 5, 6, 7, 12.5, 13, 13.5, 14, 14.5, 15.5, 30, -40, 50, 0.2, 0.1);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a joint stiffness.

Arguments

- **Comment** (Comment)

Comment that will be attached to the joint stiffness

Returns

No return value

Example

To associate comment c to the joint stiffness js:

```
js.AssociateComment(c);
```

Blank()

Description

Blanks the joint stiffness

Arguments

No arguments

Returns

No return value

Example

To blank joint stiffness js:

```
js.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the joint stiffnesses in the model.

Arguments

- **Model** (Model)

Model that all joint stiffnesses will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the joint stiffnesses in model m:

```
JointStiffness.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged joint stiffnesses in the model.

Arguments

- **Model** (Model)

Model that all the flagged joint stiffnesses will be blanked in

- **flag** (Flag)

Flag set on the joint stiffnesses that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the joint stiffnesses in model m flagged with f:

```
JointStiffness.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the joint stiffness is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if joint stiffness js is blanked:

```
if (js.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse joint stiffness js:

```
js.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the joint stiffness.

Arguments

- **flag** (Flag)

Flag to clear on the joint stiffness

Returns

No return value

Example

To clear flag f for joint stiffness js:

```
js.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the joint stiffness. The target include of the copied joint stiffness can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

JointStiffness object

Return type

JointStiffness

Example

To copy joint stiffness js into joint stiffness z:

```
var z = js.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a joint stiffness

Arguments

- **Model** (Model)

Model that the joint stiffness will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

JointStiffness object (or null if not made).

Return type

JointStiffness

Example

To start creating a joint stiffness in model m:

```
var js = JointStiffness.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a joint stiffness.

Arguments

- **Comment** (Comment)

Comment that will be detached from the joint stiffness

Returns

No return value

Example

To detach comment c from the joint stiffness js:

```
js.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit joint stiffness js:

```
js.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for joint stiffness with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The joint stiffness object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for joint stiffness js:

```
js.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for joint stiffness js:

```
js.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for joint stiffness. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for joint stiffness js:

```
js.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first joint stiffness in the model.

Arguments

- **Model** (Model)

Model to get first joint stiffness in

Returns

JointStiffness object (or null if there are no joint stiffnesses in the model).

Return type

JointStiffness

Example

To get the first joint stiffness in model m:

```
var js = JointStiffness.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free joint stiffness label in the model. Also see JointStiffness.LastFreeLabel(), JointStiffness.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free joint stiffness label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

JointStiffness label.

Return type

Number

Example

To get the first free joint stiffness label in model m:

```
var label = JointStiffness.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the joint stiffnesses in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all joint stiffnesses will be flagged in

- **flag** (Flag)

Flag to set on the joint stiffnesses

Returns

No return value

Example

To flag all of the joint stiffnesses with flag f in model m:

```
JointStiffness.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the joint stiffness is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the joint stiffness

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if joint stiffness js has flag f set on it:

```
if (js.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each joint stiffness in the model.
Note that ForEach has been designed to make looping over joint stiffnesses as fast as possible and so has some limitations.
Firstly, a single temporary JointStiffness object is created and on each function call it is updated with the current joint stiffness data. This means that you should not try to store the JointStiffness object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new joint stiffnesses inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all joint stiffnesses are in

- **func** (function)

Function to call for each joint stiffness

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the joint stiffnesses in model m:

```
JointStiffness.ForEach(m, test);
function test(js)
{
// js is JointStiffness object
}
```

To call function test for all of the joint stiffnesses in model m with optional object:

```
var data = { x:0, y:0 };
JointStiffness.ForEach(m, test, data);
function test(js, extra)
{
// js is JointStiffness object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of JointStiffness objects or properties for all of the joint stiffnesses in a model in PRIMER. If the optional property argument is not given then an array of JointStiffness objects is returned. If the property argument is given, that property value for each joint stiffness is returned in the array instead of a JointStiffness object

Arguments

- **Model** (Model)

Model to get joint stiffnesses from

- **property (optional)** (string)

Name for property to get for all joint stiffnesses in the model

Returns

Array of JointStiffness objects or properties

Return type

Array

Example

To make an array of JointStiffness objects for all of the joint stiffnesses in model m:

```
var a = JointStiffness.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each joint stiffness in model m:

```
var a = JointStiffness.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a joint stiffness.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the joint stiffness js:

```
var comm_array = js.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of JointStiffness objects for all of the flagged joint stiffnesses in a model in PRIMER If the optional property argument is not given then an array of JointStiffness objects is returned. If the property argument is given, then that property value for each joint stiffness is returned in the array instead of a JointStiffness object

Arguments

- **Model** (Model)

Model to get joint stiffnesses from

- **flag** (Flag)

Flag set on the joint stiffnesses that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged joint stiffnesses in the model

Returns

Array of JointStiffness objects or properties

Return type

Array

Example

To make an array of JointStiffness objects for all of the joint stiffnesses in model m flagged with f:

```
var js = JointStiffness.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the joint stiffnesses in model m flagged with f:

```
var a = JointStiffness.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the JointStiffness object for a joint stiffness ID.

Arguments

- **Model** (Model)

Model to find the joint stiffness in

- **number** (integer)

number of the joint stiffness you want the JointStiffness object for

Returns

JointStiffness object (or null if joint stiffness does not exist).

Return type

JointStiffness

Example

To get the JointStiffness object for joint stiffness 100 in model m

```
var js = JointStiffness.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a JointStiffness property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the JointStiffness.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

joint stiffness property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if JointStiffness property js.example is a parameter:

```
Options.property_parameter_names = true;
if (js.GetParameter(js.example) ) do_something...
Options.property_parameter_names = false;
```

To check if JointStiffness property js.example is a parameter by using the GetParameter method:

```
if (js.ViewParameters().GetParameter(js.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this jstf (*CONSTRAINED_JOINT_STIFFNESS). **Note that a carriage return is not added**. See also JointStiffness.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for jstf n:

```
var key = n.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the jstf. **Note that a carriage return is not added**. See also JointStiffness.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for jstf n:

```
var cards = n.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last joint stiffness in the model.

Arguments

- **Model** (Model)

Model to get last joint stiffness in

Returns

JointStiffness object (or null if there are no joint stiffnesses in the model).

Return type

JointStiffness

Example

To get the last joint stiffness in model m:

```
var js = JointStiffness.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free joint stiffness label in the model. Also see JointStiffness.FirstFreeLabel(), JointStiffness.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free joint stiffness label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

JointStiffness label.

Return type

Number

Example

To get the last free joint stiffness label in model m:

```
var label = JointStiffness.LastFreeLabel(m);
```

Next()

Description

Returns the next joint stiffness in the model.

Arguments

No arguments

Returns

JointStiffness object (or null if there are no more joint stiffnesses in the model).

Return type

JointStiffness

Example

To get the joint stiffness in model m after joint stiffness js:

```
var js = js.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) joint stiffness label in the model. Also see JointStiffness.FirstFreeLabel(), JointStiffness.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free joint stiffness label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

JointStiffness label.

Return type

Number

Example

To get the next free joint stiffness label in model m:

```
var label = JointStiffness.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a joint stiffness.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only joint stiffnesses from that model can be picked. If the argument is a Flag then only joint stiffnesses that are flagged with *limit* can be selected. If omitted, or null, any joint stiffnesses from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

JointStiffness object (or null if not picked)

Return type

JointStiffness

Example

To pick a joint stiffness from model m giving the prompt 'Pick joint stiffness from screen':

```
var js = JointStiffness.Pick('Pick joint stiffness from screen', m);
```

Previous()

Description

Returns the previous joint stiffness in the model.

Arguments

No arguments

Returns

JointStiffness object (or null if there are no more joint stiffnesses in the model).

Return type

JointStiffness

Example

To get the joint stiffness in model m before joint stiffness js:

```
var js = js.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the joint stiffnesses in the model.

Arguments

- **Model** (Model)

Model that all joint stiffnesses will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the joint stiffnesses in model m, from 1000000:

```
JointStiffness.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged joint stiffnesses in the model.

Arguments

- **Model** (Model)

Model that all the flagged joint stiffnesses will be renumbered in

- **flag** (Flag)

Flag set on the joint stiffnesses that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the joint stiffnesses in model m flagged with f, from 1000000:

```
JointStiffness.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select joint stiffnesses using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting joint stiffnesses

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only joint stiffnesses from that model can be selected. If the argument is a Flag then only joint stiffnesses that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any joint stiffnesses can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of joint stiffnesses selected or null if menu cancelled

Return type

Number

Example

To select joint stiffnesses from model m, flagging those selected with flag f, giving the prompt 'Select joint stiffnesses':

```
JointStiffness.Select(f, 'Select joint stiffnesses', m);
```

To select joint stiffnesses, flagging those selected with flag f but limiting selection to joint stiffnesses flagged with flag l, giving the prompt 'Select joint stiffnesses':

```
JointStiffness.Select(f, 'Select joint stiffnesses', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the joint stiffness.

Arguments

- **flag** (Flag)

Flag to set on the joint stiffness

Returns

No return value

Example

To set flag f for joint stiffness js:

```
js.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the joint stiffness. The joint stiffness will be sketched until you either call JointStiffness.Unsketch(), JointStiffness.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the joint stiffness is sketched. If omitted redraw is true. If you want to sketch several joint stiffnesses and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch joint stiffness js:

```
js.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged joint stiffnesses in the model. The joint stiffnesses will be sketched until you either call JointStiffness.Unsketch(), JointStiffness.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged joint stiffnesses will be sketched in

- **flag** (Flag)

Flag set on the joint stiffnesses that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the joint stiffnesses are sketched. If omitted redraw is true. If you want to sketch flagged joint stiffnesses several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all joint stiffnesses flagged with flag in model m:

```
JointStiffness.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of joint stiffnesses in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing joint stiffnesses should be counted. If false or omitted referenced but undefined joint stiffnesses will also be included in the total.

Returns

number of joint stiffnesses

Return type

Number

Example

To get the total number of joint stiffnesses in model m:

```
var total = JointStiffness.Total(m);
```

Unblank()

Description

Unblanks the joint stiffness

Arguments

No arguments

Returns

No return value

Example

To unblank joint stiffness js:

```
js.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the joint stiffnesses in the model.

Arguments

- **Model** (Model)

Model that all joint stiffnesses will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the joint stiffnesses in model m:

```
JointStiffness.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged joint stiffnesses in the model.

Arguments

- **Model** (Model)

Model that the flagged joint stiffnesses will be unblanked in

- **flag** (Flag)

Flag set on the joint stiffnesses that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the joint stiffnesses in model m flagged with f:

```
JointStiffness.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the joint stiffnesses in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all joint stiffnesses will be unset in

- **flag** (Flag)

Flag to unset on the joint stiffnesses

Returns

No return value

Example

To unset the flag f on all the joint stiffnesses in model m:

```
JointStiffness.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the joint stiffness.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the joint stiffness is unsketched. If omitted redraw is true. If you want to unsketch several joint stiffnesses and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch joint stiffness js:

```
js.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all joint stiffnesses.

Arguments

- **Model** (Model)

Model that all joint stiffnesses will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the joint stiffnesses are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all joint stiffnesses in model m:

```
JointStiffness.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged joint stiffnesses in the model.

Arguments

- **Model** (Model)

Model that all joint stiffnesses will be unsketched in

- **flag** (Flag)

Flag set on the joint stiffnesses that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the joint stiffnesses are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all joint stiffnesses flagged with flag in model m:

```
JointStiffness.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

JointStiffness object.

Return type

JointStiffness

Example

To check if JointStiffness property js.example is a parameter by using the JointStiffness.GetParameter() method:

```
if (js.ViewParameters().GetParameter(js.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for joint stiffness with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The joint stiffness object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for joint stiffness js:

```
js.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for joint stiffness js:

```
js.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for joint stiffness. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for joint stiffness js:

```
js.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this joint stiffness.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for joint stiffness js:

```
var xrefs = js.Xrefs();
```

---

toString()

Description

Creates a string containing the jstf data in keyword format. Note that this contains the keyword header and the keyword cards. See also JointStiffness.Keyword() and JointStiffness.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for jstf n in keyword format

```
var s = n.toString();
```

Linear class

The Linear class gives you access to define *CONSTRAINED_LINEAR cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AddRowData(nid*[integer]*, dof*[integer]*, coeff*[real]*, cid (optional)*[integer]*)
- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- GetRowData(row_index*[Integer]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- RemoveRowData(row_index*[Integer]*)
- SetFlag(flag*[Flag]*)
- SetRowData(row_index*[Integer]*, nid*[integer]*, dof*[integer]*, coeff*[real]*, cid (optional)*[integer]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()

- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

Linear constants

Name	Description
Linear.GLOBAL	CNST is *CONSTRAINED_LINEAR_GLOBAL.
Linear.LOCAL	CNST is *CONSTRAINED_LINEAR_LOCAL.

Linear properties

Name	Type	Description
exists (read only)	logical	true if constrained linear exists, false if referred to but not defined.
format	constant	The Constrained Linear option. Can be Linear.GLOBAL or Linear.LOCAL.
include	integer	The Include file number that the constrained linear is in.
lcid	integer	Linear label
model (read only)	integer	The Model number that the constrained linear is in.
total (read only)	integer	Number of degree-of-freedom cards.

Detailed Description

The Linear class allows you to create, modify, edit and manipulate *CONSTRAINED_LINEAR cards. See the documentation below for more details.

Constructor

new Linear(Model*[Model]*, format*[constant]*, lcid*[integer]*, nid*[integer]*, dof*[integer]*, coeff*[real]*, cid (optional)*[integer]*)

Description

Create a new Linear object.

Arguments

- **Model** (Model)

Model that Linear will be created in

- **format** (constant)

Specify the type of constrained linear. Can be Linear.GLOBAL or Linear.LOCAL)

- **lcid** (integer)

Linear label

- **nid** (integer)

Node id.

- **dof** (integer)

Degrees-of-Freedom.

- **coeff** (real)

Non-zero coefficient.

- **cid (optional)** (integer)

Coordinate System ID if format is Linear.LOCAL. The default value is 0.

Returns

Linear object

Return type

Linear

Example

To create a new constrained linear in model m of type LOCAL with lcid 1, nid 1, dof 3, coeff 0.5 and cid 2

```
var c_l = new Linear(m,Linear.LOCAL,1,1,3,0.5,2);
```

Details of functions

AddRowData(nid*[integer]*, dof*[integer]*, coeff*[real]*, cid (optional)*[integer]*)

Description

Used to add additional independent card 2 to the keyword. Adds this data to the end of the selected *CONSTRAINED_LINEAR

Arguments

- **nid** (integer)

Node id.

- **dof** (integer)

Degrees-of-Freedom.

- **coeff** (real)

Non-zero coefficient.

- **cid (optional)** (integer)

Coordinate System ID if format is Linear.LOCAL. The default value is 0.

Returns

No return value

Example

To add NID 10 to the keyword c_l with dof 4, coeff 1.3, cid 2:

```
c_l.AddRowData(10,4,1.3,2);
```

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a constrained linear.

Arguments

- **Comment** (Comment)

Comment that will be attached to the constrained linear

Returns

No return value

Example

To associate comment c to the constrained linear c_l:

```
c_l.AssociateComment(c);
```

Blank()

Description

Blanks the constrained linear

Arguments

No arguments

Returns

No return value

Example

To blank constrained linear c_l:

```
c_l.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the constrained linears in the model.

Arguments

- **Model** (Model)

Model that all constrained linears will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the constrained linears in model m:

```
Linear.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged constrained linears in the model.

Arguments

- **Model** (Model)

Model that all the flagged constrained linears will be blanked in

- **flag** (Flag)

Flag set on the constrained linears that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the constrained linears in model m flagged with f:

```
Linear.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the constrained linear is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if constrained linear c_l is blanked:

```
if (c_l.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse constrained linear c_l:

```
c_l.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the constrained linear.

Arguments

- **flag** (Flag)

Flag to clear on the constrained linear

Returns

No return value

Example

To clear flag f for constrained linear c_l:

```
c_l.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the constrained linear. The target include of the copied constrained linear can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

Linear object

Return type

Linear

Example

To copy constrained linear c_l into constrained linear z:

```
var z = c_l.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a constrained linear

Arguments

- **Model** (Model)

Model that the constrained linear will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

Linear object (or null if not made).

Return type

Linear

Example

To start creating a constrained linear in model m:

```
var c_l = Linear.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a constrained linear.

Arguments

- **Comment** (Comment)

Comment that will be detached from the constrained linear

Returns

No return value

Example

To detach comment c from the constrained linear c_l:

```
c_l.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit constrained linear c_l:

```
c_l.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for constrained linear with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The constrained linear object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for constrained linear c_l:

```
c_l.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for constrained linear c_l:

```
c_l.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for constrained linear. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for constrained linear c_l:

```
c_l.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first constrained linear in the model.

Arguments

- **Model** (Model)

Model to get first constrained linear in

Returns

Linear object (or null if there are no constrained linears in the model).

Return type

Linear

Example

To get the first constrained linear in model m:

```
var c_l = Linear.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free constrained linear label in the model. Also see Linear.LastFreeLabel(), Linear.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free constrained linear label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

Linear label.

Return type

Number

Example

To get the first free constrained linear label in model m:

```
var label = Linear.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the constrained linears in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all constrained linears will be flagged in

- **flag** (Flag)

Flag to set on the constrained linears

Returns

No return value

Example

To flag all of the constrained linears with flag f in model m:

```
Linear.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the constrained linear is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the constrained linear

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if constrained linear c_l has flag f set on it:

```
if (c_l.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each constrained linear in the model.
Note that ForEach has been designed to make looping over constrained linears as fast as possible and so has some limitations.
Firstly, a single temporary Linear object is created and on each function call it is updated with the current constrained linear data. This means that you should not try to store the Linear object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new constrained linears inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all constrained linears are in

- **func** (function)

Function to call for each constrained linear

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the constrained linears in model m:

```
Linear.ForEach(m, test);
function test(c_l)
{
// c_l is Linear object
}
```

To call function test for all of the constrained linears in model m with optional object:

```
var data = { x:0, y:0 };
Linear.ForEach(m, test, data);
function test(c_l, extra)
{
// c_l is Linear object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Linear objects or properties for all of the constrained linears in a model in PRIMER. If the optional property argument is not given then an array of Linear objects is returned. If the property argument is given, that property value for each constrained linear is returned in the array instead of a Linear object

Arguments

- **Model** (Model)

Model to get constrained linears from

- **property (optional)** (string)

Name for property to get for all constrained linears in the model

Returns

Array of Linear objects or properties

Return type

Array

Example

To make an array of Linear objects for all of the constrained linears in model m:

```
var a = Linear.GetAll(m);
```

To return an array containing the value of property ’foo’ (for example ’x’ for a node) for each constrained linear in model m:

```
var a = Linear.GetAll(m, ’foo’);
```

GetComments()

Description

Extracts the comments associated to a constrained linear.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the constrained linear c_l:

```
var comm_array = c_l.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of Linear objects for all of the flagged constrained linears in a model in PRIMER If the optional property argument is not given then an array of Linear objects is returned. If the property argument is given, then that property value for each constrained linear is returned in the array instead of a Linear object

Arguments

- **Model** (Model)

Model to get constrained linears from

- **flag** (Flag)

Flag set on the constrained linears that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged constrained linears in the model

Returns

Array of Linear objects or properties

Return type

Array

Example

To make an array of Linear objects for all of the constrained linears in model m flagged with f:

```
var c_l = Linear.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the constrained linears in model m flagged with f:

```
var a = Linear.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the Linear object for a constrained linear ID.

Arguments

- **Model** (Model)

Model to find the constrained linear in

- **number** (integer)

number of the constrained linear you want the Linear object for

Returns

Linear object (or null if constrained linear does not exist).

Return type

Linear

Example

To get the Linear object for constrained linear 100 in model m

```
var c_l = Linear.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a Linear property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the Linear.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

constrained linear property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if Linear property c_l.example is a parameter:

```
Options.property_parameter_names = true;
if (c_l.GetParameter(c_l.example) ) do_something...
Options.property_parameter_names = false;
```

To check if Linear property c_l.example is a parameter by using the GetParameter method:

```
if (c_l.ViewParameters().GetParameter(c_l.example) ) do_something...
```

GetRowData(row_index*[Integer]*)

Description

Returns independent card 2 for the selected row of the *CONSTRAINED_LINEAR.

Arguments

- **row_index** (Integer)

The row index of the data to return. **Note that indices start at 0, not 1**.
0 <= row_index < Linear.total

Returns

Array containing data.

Return type

Array

Example

To loop over all the lines of the keyword for c_l:

```
for (i=0; i<c_l.total; i++)
        var data = c_l.GetRowData(i);
```

Keyword()

Description

Returns the keyword for this Linear (*constrained_linear). **Note that a carriage return is not added**. See also Linear.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for Linear c_l:

```
var key = c_l.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the Linear. **Note that a carriage return is not added**. See also Linear.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for Linear c_l:

```
var cards = c_l.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last constrained linear in the model.

Arguments

- **Model** (Model)

Model to get last constrained linear in

Returns

Linear object (or null if there are no constrained linears in the model).

Return type

Linear

Example

To get the last constrained linear in model m:

```
var c_l = Linear.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free constrained linear label in the model. Also see Linear.FirstFreeLabel(), Linear.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free constrained linear label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

Linear label.

Return type

Number

Example

To get the last free constrained linear label in model m:

```
var label = Linear.LastFreeLabel(m);
```

Next()

Description

Returns the next constrained linear in the model.

Arguments

No arguments

Returns

Linear object (or null if there are no more constrained linears in the model).

Return type

Linear

Example

To get the constrained linear in model m after constrained linear c_l:

```
var c_l = c_l.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) constrained linear label in the model. Also see Linear.FirstFreeLabel(), Linear.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free constrained linear label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

Linear label.

Return type

Number

Example

To get the next free constrained linear label in model m:

```
var label = Linear.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a constrained linear.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only constrained linears from that model can be picked. If the argument is a Flag then only constrained linears that are flagged with *limit* can be selected. If omitted, or null, any constrained linears from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

Linear object (or null if not picked)

Return type

Linear

Example

To pick a constrained linear from model m giving the prompt 'Pick constrained linear from screen':

```
var c_l = Linear.Pick('Pick constrained linear from screen', m);
```

Previous()

Description

Returns the previous constrained linear in the model.

Arguments

No arguments

Returns

Linear object (or null if there are no more constrained linears in the model).

Return type

Linear

Example

To get the constrained linear in model m before constrained linear c_l:

```
var c_l = c_l.Previous();
```

RemoveRowData(row_index*[Integer]*)

Description

Removes an independent card 2 for the selected row on the *CONSTRAINED_LINEAR.

Arguments

- **row_index** (Integer)

The row index of the data to return. **Note that indices start at 0, not 1**.
0 <= row_index < Linear.total

Returns

No return value.

Example

To remove row 2 for c_l:

```
c_l.RemoveRowData(1);
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the constrained linears in the model.

Arguments

- **Model** (Model)

Model that all constrained linears will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the constrained linears in model m, from 1000000:

```
Linear.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged constrained linears in the model.

Arguments

- **Model** (Model)

Model that all the flagged constrained linears will be renumbered in

- **flag** (Flag)

Flag set on the constrained linears that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the constrained linears in model m flagged with f, from 1000000:

```
Linear.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select constrained linears using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting constrained linears

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only constrained linears from that model can be selected. If the argument is a Flag then only constrained linears that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any constrained linears can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of constrained linears selected or null if menu cancelled

Return type

Number

Example

To select constrained linears from model m, flagging those selected with flag f, giving the prompt 'Select constrained linears':

```
Linear.Select(f, 'Select constrained linears', m);
```

To select constrained linears, flagging those selected with flag f but limiting selection to constrained linears flagged with flag l, giving the prompt 'Select constrained linears':

```
Linear.Select(f, 'Select constrained linears', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the constrained linear.

Arguments

- **flag** (Flag)

Flag to set on the constrained linear

Returns

No return value

Example

To set flag f for constrained linear c_l:

```
c_l.SetFlag(f);
```

SetRowData(row_index*[Integer]*, nid*[integer]*, dof*[integer]*, coeff*[real]*, cid (optional)*[integer]*)

Description

Used to reset values in already existing card 2 in the selected row of *CONSTRAINED_LINEAR

Arguments

- **row_index** (Integer)

The row index of the data to return. **Note that indices start at 0, not 1**.
0 <= row_index < Linear.total

- **nid** (integer)

Node id.

- **dof** (integer)

Degrees-of-Freedom.

- **coeff** (real)

Non-zero coefficient.

- **cid (optional)** (integer)

Coordinate System ID if format is Linear.LOCAL. The default value is 0.

Returns

No return value

Example

To reset the values of row 3 of the keyword with NID 11, dof 2, coeff 3.2, cid 4:

```
c_l.SetRowData(2,11,2,3.2,4);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the constrained linear. The constrained linear will be sketched until you either call Linear.Unsketch(), Linear.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the constrained linear is sketched. If omitted redraw is true. If you want to sketch several constrained linears and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch constrained linear c_l:

```
c_l.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged constrained linears in the model. The constrained linears will be sketched until you either call Linear.Unsketch(), Linear.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged constrained linears will be sketched in

- **flag** (Flag)

Flag set on the constrained linears that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the constrained linears are sketched. If omitted redraw is true. If you want to sketch flagged constrained linears several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all constrained linears flagged with flag in model m:

```
Linear.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of constrained linears in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing constrained linears should be counted. If false or omitted referenced but undefined constrained linears will also be included in the total.

Returns

number of constrained linears

Return type

Number

Example

To get the total number of constrained linears in model m:

```
var total = Linear.Total(m);
```

Unblank()

Description

Unblanks the constrained linear

Arguments

No arguments

Returns

No return value

Example

To unblank constrained linear c_l:

```
c_l.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the constrained linears in the model.

Arguments

- **Model** (Model)

Model that all constrained linears will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the constrained linears in model m:

```
Linear.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged constrained linears in the model.

Arguments

- **Model** (Model)

Model that the flagged constrained linears will be unblanked in

- **flag** (Flag)

Flag set on the constrained linears that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the constrained linears in model m flagged with f:

```
Linear.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the constrained linears in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all constrained linears will be unset in

- **flag** (Flag)

Flag to unset on the constrained linears

Returns

No return value

Example

To unset the flag f on all the constrained linears in model m:

```
Linear.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the constrained linear.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the constrained linear is unsketched. If omitted redraw is true. If you want to unsketch several constrained linears and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch constrained linear c_l:

```
c_l.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all constrained linears.

Arguments

- **Model** (Model)

Model that all constrained linears will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the constrained linears are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all constrained linears in model m:

```
Linear.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged constrained linears in the model.

Arguments

- **Model** (Model)

Model that all constrained linears will be unsketched in

- **flag** (Flag)

Flag set on the constrained linears that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the constrained linears are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all constrained linears flagged with flag in model m:

```
Linear.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

Linear object.

Return type

Linear

Example

To check if Linear property c_l.example is a parameter by using the Linear.GetParameter() method:

```
if (c_l.ViewParameters().GetParameter(c_l.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for constrained linear with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description

autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The constrained linear object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for constrained linear c_l:

```
c_l.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for constrained linear c_l:

```
c_l.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for constrained linear. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for constrained linear c_l:

```
c_l.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this constrained linear.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for constrained linear c_l:

```
var xrefs = c_l.Xrefs();
```

---

toString()

Description

Creates a string containing the Linear data in keyword format. Note that this contains the keyword header and the keyword cards. See also Linear.Keyword() and Linear.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for Linear c_l in keyword format

```
var s = c_l.toString();
```

NodalRigidBody (Nrb) class

The NodalRigidBody class gives you access to define nodal rigid body cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- ExtractColour()
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)

- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

NodalRigidBody properties

Name	Type	Description
cid	integer	Coordinate system ID
cmo	integer	Centre of mass option
colour	Colour	The colour of the nrb
con1	integer	First retraint parameter
con2	integer	Second retraint parameter
drflag	integer	Displacement release flag
exists (read only)	logical	true if nodal rigid body exists, false if referred to but not defined.
idthrm	integer	Flag for the thermal constraint
include	integer	The Include file number that the nodal rigid body is in.
inertia	logical	Flag to turn on or off _INERTIA option
iprt	integer	Print flag
ixx	real	Ixx component of inertia tensor
ixy	real	Ixy component of inertia tensor
ixz	real	Ixz component of inertia tensor
iyy	real	Iyy component of inertia tensor
iyz	real	Iyz component of inertia tensor
izz	real	Izz component of inertia tensor
label	integer	NodalRigidBody ID of the NRB. Also see the pid property which is an alternative name for this.
model (read only)	integer	The Model number that the nodal rigid body is in.
nodeid	integer	Optional node point
nsid	integer	Nodal set ID
override	logical	Flag to turn on or off _OVERRIDE option
pid	integer	NodalRigidBody ID of the NRB. Also see the label property which is an alternative name for this.
pnode	integer	Optional nodal point
rrflag	integer	Rotation release flag
spc	logical	Flag to turn on or off _SPC option
spcnid	integer	Node set ID for SPCs
thermal	logical	Flag to turn on or off _THERMAL option
tm	real	Translational mass
vrx	real	X rigid body rotational velocity
vry	real	Y rigid body rotational velocity
vrz	real	Z rigid body rotational velocity

vtx	real	X rigid body translational velocity
vty	real	Y rigid body translational velocity
vtz	real	Z rigid body translational velocity
xc	real	X coordinate centre of mass
xl	real	X coordinate of local x axis
xlip	real	X coordinate of local in plane vector
xspc	real	X coordinate where the constraints act
yc	real	Y coordinate centre of mass
yl	real	Y coordinate of local x axis
ylip	real	Y coordinate of local in plane vector
yspc	real	Y coordinate where the constraints act
zc	real	Z coordinate centre of mass
zl	real	Z coordinate of local x axis
zlip	real	Z coordinate of local in plane vector
zspc	real	Z coordinate where the constraints act

Detailed Description

The NodalRigidBody class allows you to create, modify, edit and manipulate nodal rigid body cards. See the documentation below for more details.

For convenience "Nrb" can also be used as the class name instead of "NodalRigidBody".

Constructor

new NodalRigidBody(Model*[Model]*, nsid*[integer]*, pid (optional)*[integer]*, cid (optional)*[integer]*, pnode (optional)*[integer]*, iprt (optional)*[integer]*, drflag (optional)*[integer]*, rrflag (optional)*[integer]*)

Description

Create a new NodalRigidBody object.

Arguments

- **Model** (Model)

Model that nrb will be created in

- **nsid** (integer)

Nodal set ID

- **pid (optional)** (integer)

NodalRigidBody ID of the NRB. Also see the label property which is an alternative name for this.

- **cid (optional)** (integer)

Coordinate system ID

- **pnode (optional)** (integer)

Optional nodal point

- **iprt (optional)** (integer)

Print flag

- **drflag (optional)** (integer)

Displacement release flag

- **rrflag (optional)** (integer)

Rotation release flag

Returns

NodalRigidBody object

Return type

NodalRigidBody

Example

To create a new nrb in model m with label 200, using node set 50

```
var v = new NodalRigidBody(m, 50, 200);
```

Details of functions

AssociateComment(Comment[*Comment*])

Description

Associates a comment with a nodal rigid body.

Arguments

- **Comment** (Comment)

Comment that will be attached to the nodal rigid body

Returns

No return value

Example

To associate comment c to the nodal rigid body nrb:

```
nrb.AssociateComment(c);
```

Blank()

Description

Blanks the nodal rigid body

Arguments

No arguments

Returns

No return value

Example

To blank nodal rigid body nrb:

```
nrb.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the nodal rigid bodies in the model.

Arguments

- **Model** (Model)

Model that all nodal rigid bodies will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the nodal rigid bodies in model m:

```
NodalRigidBody.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged nodal rigid bodies in the model.

Arguments

- **Model** (Model)

Model that all the flagged nodal rigid bodies will be blanked in

- **flag** (Flag)

Flag set on the nodal rigid bodies that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the nodal rigid bodies in model m flagged with f:

```
NodalRigidBody.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the nodal rigid body is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if nodal rigid body nrb is blanked:

```
if (nrb.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse nodal rigid body nrb:

```
nrb.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the nodal rigid body.

Arguments

- **flag** (Flag)

Flag to clear on the nodal rigid body

Returns

No return value

Example

To clear flag f for nodal rigid body nrb:

```
nrb.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the nodal rigid body. The target include of the copied nodal rigid body can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

NodalRigidBody object

Return type

NodalRigidBody

Example

To copy nodal rigid body nrb into nodal rigid body z:

```
var z = nrb.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a nodal rigid body

Arguments

- **Model** (Model)

Model that the nodal rigid body will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

NodalRigidBody object (or null if not made).

Return type

NodalRigidBody

Example

To start creating a nodal rigid body in model m:

```
var nrb = NodalRigidBody.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a nodal rigid body.

Arguments

- **Comment** (Comment)

Comment that will be detached from the nodal rigid body

Returns

No return value

Example

To detach comment c from the nodal rigid body nrb:

```
nrb.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit nodal rigid body nrb:

```
nrb.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for nodal rigid body with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The nodal rigid body object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for nodal rigid body nrb:

```
nrb.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for nodal rigid body nrb:

```
nrb.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for nodal rigid body. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for nodal rigid body nrb:

```
nrb.Error("My custom error");
```

ExtractColour()

Description

Extracts the **actual** colour used for nodal rigid body.
By default in PRIMER many entities such as elements get their colour automatically from the part that they are in. PRIMER cycles through 13 default colours based on the label of the entity. In this case the nodal rigid body colour property will return the value Colour.PART instead of the actual colour. This method will return the actual colour which is used for drawing the nodal rigid body.

Arguments

No arguments

Returns

colour value (integer)

Return type

Number

Example

To return the colour used for drawing nodal rigid body nrb:

```
var colour = nrb.ExtractColour();
```

First(Model*[Model]*) [static]

Description

Returns the first nodal rigid body in the model.

Arguments

- **Model** (Model)

Model to get first nodal rigid body in

Returns

NodalRigidBody object (or null if there are no nodal rigid bodies in the model).

Return type

NodalRigidBody

Example

To get the first nodal rigid body in model m:

```
var nrb = NodalRigidBody.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free nodal rigid body label in the model. Also see NodalRigidBody.LastFreeLabel(), NodalRigidBody.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free nodal rigid body label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

NodalRigidBody label.

Return type

Number

Example

To get the first free nodal rigid body label in model m:

```
var label = NodalRigidBody.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the nodal rigid bodies in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all nodal rigid bodies will be flagged in

- **flag** (Flag)

Flag to set on the nodal rigid bodies

Returns

No return value

Example

To flag all of the nodal rigid bodies with flag f in model m:

```
NodalRigidBody.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the nodal rigid body is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the nodal rigid body

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if nodal rigid body nrb has flag f set on it:

```
if (nrb.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each nodal rigid body in the model.
Note that ForEach has been designed to make looping over nodal rigid bodies as fast as possible and so has some limitations.
Firstly, a single temporary NodalRigidBody object is created and on each function call it is updated with the current nodal rigid body data. This means that you should not try to store the NodalRigidBody object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new nodal rigid bodies inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all nodal rigid bodies are in

- **func** (function)

Function to call for each nodal rigid body

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the nodal rigid bodies in model m:

```
NodalRigidBody.ForEach(m, test);
function test(nrb)
{
// nrb is NodalRigidBody object
}
```

To call function test for all of the nodal rigid bodies in model m with optional object:

```
var data = { x:0, y:0 };
NodalRigidBody.ForEach(m, test, data);
function test(nrb, extra)
{
// nrb is NodalRigidBody object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of NodalRigidBody objects or properties for all of the nodal rigid bodies in a model in PRIMER. If the optional property argument is not given then an array of NodalRigidBody objects is returned. If the property argument is given, that property value for each nodal rigid body is returned in the array instead of a NodalRigidBody object

Arguments

- **Model** (Model)

Model to get nodal rigid bodies from

- **property (optional)** (string)

Name for property to get for all nodal rigid bodies in the model

Returns

Array of NodalRigidBody objects or properties

Return type

Array

Example

To make an array of NodalRigidBody objects for all of the nodal rigid bodies in model m:

```
var a = NodalRigidBody.GetAll(m);
```

To return an array containing the value of property ’foo’ (for example ’x’ for a node) for each nodal rigid body in model m:

```
var a = NodalRigidBody.GetAll(m, ’foo’);
```

GetComments()

Description

Extracts the comments associated to a nodal rigid body.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the nodal rigid body nrb:

```
var comm_array = nrb.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of NodalRigidBody objects for all of the flagged nodal rigid bodies in a model in PRIMER If the optional property argument is not given then an array of NodalRigidBody objects is returned. If the property argument is given, then that property value for each nodal rigid body is returned in the array instead of a NodalRigidBody object

Arguments

- **Model** (Model)

Model to get nodal rigid bodies from

- **flag** (Flag)

Flag set on the nodal rigid bodies that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged nodal rigid bodies in the model

Returns

Array of NodalRigidBody objects or properties

Return type

Array

Example

To make an array of NodalRigidBody objects for all of the nodal rigid bodies in model m flagged with f:

```
var nrb = NodalRigidBody.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the nodal rigid bodies in model m flagged with f:

```
var a = NodalRigidBody.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the NodalRigidBody object for a nodal rigid body ID.

Arguments

- **Model** (Model)

Model to find the nodal rigid body in

- **number** (integer)

number of the nodal rigid body you want the NodalRigidBody object for

Returns

NodalRigidBody object (or null if nodal rigid body does not exist).

Return type

NodalRigidBody

Example

To get the NodalRigidBody object for nodal rigid body 100 in model m

```
var nrb = NodalRigidBody.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a NodalRigidBody property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the NodalRigidBody.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

nodal rigid body property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if NodalRigidBody property nrb.example is a parameter:

```
Options.property_parameter_names = true;
if (nrb.GetParameter(nrb.example) ) do_something...
Options.property_parameter_names = false;
```

To check if NodalRigidBody property nrb.example is a parameter by using the GetParameter method:

```
if (nrb.ViewParameters().GetParameter(nrb.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this nrb (*CONSTRAINED_NODAL_RIGID_BODY_xxxx). **Note that a carriage return is not added**. See also NodalRigidBody.KeywordCards()

Arguments

No arguments

Returns
string containing the keyword.

Return type
String

Example
To get the keyword for nrb n:
```
var key = n.Keyword();
```

KeywordCards()

Description
Returns the keyword cards for the nrb. **Note that a carriage return is not added**. See also NodalRigidBody.Keyword()

Arguments
No arguments

Returns
string containing the cards.

Return type
String

Example
To get the cards for nrb n:
```
var cards = n.KeywordCards();
```

Last(Model[*Model*]) [static]

Description
Returns the last nodal rigid body in the model.

Arguments
- **Model** (Model)

Model to get last nodal rigid body in

Returns
NodalRigidBody object (or null if there are no nodal rigid bodies in the model).

Return type
NodalRigidBody

Example
To get the last nodal rigid body in model m:
```
var nrb = NodalRigidBody.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free nodal rigid body label in the model. Also see NodalRigidBody.FirstFreeLabel(), NodalRigidBody.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free nodal rigid body label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

NodalRigidBody label.

Return type

Number

Example

To get the last free nodal rigid body label in model m:

```
var label = NodalRigidBody.LastFreeLabel(m);
```

Next()

Description

Returns the next nodal rigid body in the model.

Arguments

No arguments

Returns

NodalRigidBody object (or null if there are no more nodal rigid bodies in the model).

Return type

NodalRigidBody

Example

To get the nodal rigid body in model m after nodal rigid body nrb:

```
var nrb = nrb.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) nodal rigid body label in the model. Also see NodalRigidBody.FirstFreeLabel(), NodalRigidBody.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free nodal rigid body label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

NodalRigidBody label.

Return type

Number

Example

To get the next free nodal rigid body label in model m:

```
var label = NodalRigidBody.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a nodal rigid body.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only nodal rigid bodies from that model can be picked. If the argument is a Flag then only nodal rigid bodies that are flagged with *limit* can be selected. If omitted, or null, any nodal rigid bodies from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

NodalRigidBody object (or null if not picked)

Return type

NodalRigidBody

Example

To pick a nodal rigid body from model m giving the prompt 'Pick nodal rigid body from screen':

```
var nrb = NodalRigidBody.Pick('Pick nodal rigid body from screen', m);
```

Previous()

Description

Returns the previous nodal rigid body in the model.

Arguments

No arguments

Returns

NodalRigidBody object (or null if there are no more nodal rigid bodies in the model).

Return type

NodalRigidBody

Example

To get the nodal rigid body in model m before nodal rigid body nrb:

```
var nrb = nrb.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the nodal rigid bodies in the model.

Arguments

- **Model** (Model)

Model that all nodal rigid bodies will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the nodal rigid bodies in model m, from 1000000:

```
NodalRigidBody.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged nodal rigid bodies in the model.

Arguments

- **Model** (Model)

Model that all the flagged nodal rigid bodies will be renumbered in

- **flag** (Flag)

Flag set on the nodal rigid bodies that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the nodal rigid bodies in model m flagged with f, from 1000000:

```
NodalRigidBody.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select nodal rigid bodies using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting nodal rigid bodies

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only nodal rigid bodies from that model can be selected. If the argument is a Flag then only nodal rigid bodies that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any nodal rigid bodies can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of nodal rigid bodies selected or null if menu cancelled

Return type

Number

Example

To select nodal rigid bodies from model m, flagging those selected with flag f, giving the prompt 'Select nodal rigid bodies':

```
NodalRigidBody.Select(f, 'Select nodal rigid bodies', m);
```

To select nodal rigid bodies, flagging those selected with flag f but limiting selection to nodal rigid bodies flagged with flag l, giving the prompt 'Select nodal rigid bodies':

```
NodalRigidBody.Select(f, 'Select nodal rigid bodies', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the nodal rigid body.

Arguments

- **flag** (Flag)

Flag to set on the nodal rigid body

Returns

No return value

Example

To set flag f for nodal rigid body nrb:

```
nrb.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the nodal rigid body. The nodal rigid body will be sketched until you either call NodalRigidBody.Unsketch(), NodalRigidBody.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the nodal rigid body is sketched. If omitted redraw is true. If you want to sketch several nodal rigid bodies and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch nodal rigid body nrb:

```
nrb.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged nodal rigid bodies in the model. The nodal rigid bodies will be sketched until you either call NodalRigidBody.Unsketch(), NodalRigidBody.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged nodal rigid bodies will be sketched in

- **flag** (Flag)

Flag set on the nodal rigid bodies that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the nodal rigid bodies are sketched. If omitted redraw is true. If you want to sketch flagged nodal rigid bodies several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all nodal rigid bodies flagged with flag in model m:

```
NodalRigidBody.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of nodal rigid bodies in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing nodal rigid bodies should be counted. If false or omitted referenced but undefined nodal rigid bodies will also be included in the total.

Returns

number of nodal rigid bodies

Return type

Number

Example

To get the total number of nodal rigid bodies in model m:

```
var total = NodalRigidBody.Total(m);
```

Unblank()

Description

Unblanks the nodal rigid body

Arguments

No arguments

Returns

No return value

Example

To unblank nodal rigid body nrb:

```
nrb.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the nodal rigid bodies in the model.

Arguments

- **Model** (Model)

Model that all nodal rigid bodies will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the nodal rigid bodies in model m:

```
NodalRigidBody.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged nodal rigid bodies in the model.

Arguments

- **Model** (Model)

Model that the flagged nodal rigid bodies will be unblanked in

- **flag** (Flag)

Flag set on the nodal rigid bodies that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the nodal rigid bodies in model m flagged with f:

```
NodalRigidBody.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the nodal rigid bodies in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all nodal rigid bodies will be unset in

- **flag** (Flag)

Flag to unset on the nodal rigid bodies

Returns

No return value

Example

To unset the flag f on all the nodal rigid bodies in model m:

```
NodalRigidBody.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the nodal rigid body.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the nodal rigid body is unsketched. If omitted redraw is true. If you want to unsketch several nodal rigid bodies and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch nodal rigid body nrb:

```
nrb.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all nodal rigid bodies.

Arguments

- **Model** (Model)

Model that all nodal rigid bodies will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the nodal rigid bodies are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all nodal rigid bodies in model m:

```
NodalRigidBody.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged nodal rigid bodies in the model.

Arguments

- **Model** (Model)

Model that all nodal rigid bodies will be unsketched in

- **flag** (Flag)

Flag set on the nodal rigid bodies that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the nodal rigid bodies are unsketched. If omitted redraw is true. If you want to

unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all nodal rigid bodies flagged with flag in model m:

```
NodalRigidBody.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

NodalRigidBody object.

Return type

NodalRigidBody

Example

To check if NodalRigidBody property nrb.example is a parameter by using the NodalRigidBody.GetParameter() method:

```
if (nrb.ViewParameters().GetParameter(nrb.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for nodal rigid body with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The nodal rigid body object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...

autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for nodal rigid body nrb:

```
nrb.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for nodal rigid body nrb:

```
nrb.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for nodal rigid body. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for nodal rigid body nrb:

```
nrb.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this nodal rigid body.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for nodal rigid body nrb:

```
var xrefs = nrb.Xrefs();
```

---

toString()

Description

Creates a string containing the nrb data in keyword format. Note that this contains the keyword header and the keyword cards. See also NodalRigidBody.Keyword() and NodalRigidBody.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for nrb n in keyword format

```
var s = n.toString();
```

NodeSet class

The NodeSet class gives you access to constrained node set cards in PRIMER, **not** set node cards. For access to set node cards, refer to the Set class. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- ExtractColour()
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()

- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

NodeSet properties

Name	Type	Description
cnsid	integer	Constrained node set number (identical to label).
colour	Colour	The colour of the node set
dof	integer	Degree of freedom.
exists (read only)	logical	true if node set exists, false if referred to but not defined.
id	logical	true if _ID option is set, false if not
include	integer	The Include file number that the node set is in.
label	integer	Constrained node set number.
model (read only)	integer	The Model number that the node set is in.
nsid	integer	Set Node ID.
tf	real	Failure time.

Detailed Description

The NodeSet class allows you to create, modify, edit and manipulate constrained node set cards. See the documentation below for more details.

Constructor

new NodeSet(Model*[Model]*, nsid*[integer]*, dof*[integer]*, tf*[real]*, label (optional)*[integer]*)

Description

Create a new NodeSet object.

Arguments

- **Model** (Model)

Model that constrained node set will be created in

- **nsid** (integer)

Set Node ID.

- **dof** (integer)

Degree of freedom.

- **tf** (real)

Failure time.

- **label (optional)** (integer)

Constrained node set number.

Returns

NodeSet object

Return type

NodeSet

Example

To create a new constrained node set 500 in model m, of type SET, with node set 9, degree of freedom 1 and failure time 1000

```
var n = new NodeSet(m, 9, 1, 1000, 500);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a node set.

Arguments

- **Comment** (Comment)

Comment that will be attached to the node set

Returns

No return value

Example

To associate comment c to the node set ns:

```
ns.AssociateComment(c);
```

Blank()

Description

Blanks the node set

Arguments

No arguments

Returns

No return value

Example

To blank node set ns:

```
ns.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the node sets in the model.

Arguments

- **Model** (Model)

Model that all node sets will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the node sets in model m:

```
NodeSet.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged node sets in the model.

Arguments

- **Model** (Model)

Model that all the flagged node sets will be blanked in

- **flag** (Flag)

Flag set on the node sets that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the node sets in model m flagged with f:

```
NodeSet.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the node set is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if node set ns is blanked:

```
if (ns.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse node set ns:

```
ns.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the node set.

Arguments

- **flag** (Flag)

Flag to clear on the node set

Returns

No return value

Example

To clear flag f for node set ns:

```
ns.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the node set. The target include of the copied node set can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

NodeSet object

Return type

NodeSet

Example

To copy node set ns into node set z:

```
var z = ns.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a node set

Arguments

- **Model** (Model)

Model that the node set will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

NodeSet object (or null if not made).

Return type

NodeSet

Example

To start creating a node set in model m:

```
var ns = NodeSet.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a node set.

Arguments

- **Comment** (Comment)

Comment that will be detached from the node set

Returns

No return value

Example

To detach comment c from the node set ns:

```
ns.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit node set ns:

```
ns.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for node set with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The node set object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for node set ns:

```
ns.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for node set ns:

```
ns.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for node set. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for node set ns:

```
ns.Error("My custom error");
```

ExtractColour()

Description

Extracts the **actual** colour used for node set.
By default in PRIMER many entities such as elements get their colour automatically from the part that they are in. PRIMER cycles through 13 default colours based on the label of the entity. In this case the node set colour property will return the value Colour.PART instead of the actual colour. This method will return the actual colour which is used for drawing the node set.

Arguments

No arguments

Returns

colour value (integer)

Return type

Number

Example

To return the colour used for drawing node set ns:

```
var colour = ns.ExtractColour();
```

First(Model*[Model]*) [static]

Description

Returns the first node set in the model.

Arguments

- **Model** (Model)

Model to get first node set in

Returns

NodeSet object (or null if there are no node sets in the model).

Return type

NodeSet

Example

To get the first node set in model m:

```
var ns = NodeSet.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free node set label in the model. Also see NodeSet.LastFreeLabel(), NodeSet.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free node set label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

NodeSet label.

Return type

Number

Example

To get the first free node set label in model m:

```
var label = NodeSet.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the node sets in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all node sets will be flagged in

- **flag** (Flag)

Flag to set on the node sets

Returns

No return value

Example

To flag all of the node sets with flag f in model m:

```
NodeSet.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the node set is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the node set

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if node set ns has flag f set on it:

```
if (ns.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each node set in the model.
Note that ForEach has been designed to make looping over node sets as fast as possible and so has some limitations.
Firstly, a single temporary NodeSet object is created and on each function call it is updated with the current node set data. This means that you should not try to store the NodeSet object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new node sets inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all node sets are in

- **func** (function)

Function to call for each node set

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the node sets in model m:

```
NodeSet.ForEach(m, test);
function test(ns)
{
// ns is NodeSet object
}
```

To call function test for all of the node sets in model m with optional object:

```
var data = { x:0, y:0 };
NodeSet.ForEach(m, test, data);
function test(ns, extra)
{
// ns is NodeSet object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of NodeSet objects or properties for all of the node sets in a model in PRIMER. If the optional property argument is not given then an array of NodeSet objects is returned. If the property argument is given, that property value for each node set is returned in the array instead of a NodeSet object

Arguments

- **Model** (Model)

Model to get node sets from

- **property (optional)** (string)

Name for property to get for all node sets in the model

Returns

Array of NodeSet objects or properties

Return type

Array

Example

To make an array of NodeSet objects for all of the node sets in model m:

```
var a = NodeSet.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each node set in model m:

```
var a = NodeSet.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a node set.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the node set ns:

```
var comm_array = ns.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of NodeSet objects for all of the flagged node sets in a model in PRIMER If the optional property argument is not given then an array of NodeSet objects is returned. If the property argument is given, then that property value for each node set is returned in the array instead of a NodeSet object

Arguments

- **Model** (Model)

Model to get node sets from

- **flag** (Flag)

Flag set on the node sets that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged node sets in the model

Returns

Array of NodeSet objects or properties

Return type

Array

Example

To make an array of NodeSet objects for all of the node sets in model m flagged with f:

```
var ns = NodeSet.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the node sets in model m flagged with f:

```
var a = NodeSet.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the NodeSet object for a node set ID.

Arguments

- **Model** (Model)

Model to find the node set in

- **number** (integer)

number of the node set you want the NodeSet object for

Returns

NodeSet object (or null if node set does not exist).

Return type

NodeSet

Example

To get the NodeSet object for node set 100 in model m

```
var ns = NodeSet.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a NodeSet property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the NodeSet.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

node set property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if NodeSet property ns.example is a parameter:

```
Options.property_parameter_names = true;
if (ns.GetParameter(ns.example) ) do_something...
Options.property_parameter_names = false;
```

To check if NodeSet property ns.example is a parameter by using the GetParameter method:

```
if (ns.ViewParameters().GetParameter(ns.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this node_set (*CONSTRAINED_NODE_SET). **Note that a carriage return is not added**. See also NodeSet.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for node_set n:

```
var key = n.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the node_set. **Note that a carriage return is not added**. See also NodeSet.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for node_set n:

```
var cards = n.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last node set in the model.

Arguments

- **Model** (Model)

Model to get last node set in

Returns

NodeSet object (or null if there are no node sets in the model).

Return type

NodeSet

Example

To get the last node set in model m:

```
var ns = NodeSet.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free node set label in the model. Also see NodeSet.FirstFreeLabel(), NodeSet.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free node set label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

NodeSet label.

Return type

Number

Example

To get the last free node set label in model m:

```
var label = NodeSet.LastFreeLabel(m);
```

Next()

Description

Returns the next node set in the model.

Arguments

No arguments

Returns

NodeSet object (or null if there are no more node sets in the model).

Return type

NodeSet

Example

To get the node set in model m after node set ns:

```
var ns = ns.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) node set label in the model. Also see NodeSet.FirstFreeLabel(), NodeSet.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free node set label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

NodeSet label.

Return type

Number

Example

To get the next free node set label in model m:

```
var label = NodeSet.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a node set.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only node sets from that model can be picked. If the argument is a Flag then only node sets that are flagged with *limit* can be selected. If omitted, or null, any node sets from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

NodeSet object (or null if not picked)

Return type

NodeSet

Example

To pick a node set from model m giving the prompt 'Pick node set from screen':

```
var ns = NodeSet.Pick('Pick node set from screen', m);
```

Previous()

Description

Returns the previous node set in the model.

Arguments

No arguments

Returns

NodeSet object (or null if there are no more node sets in the model).

Return type

NodeSet

Example

To get the node set in model m before node set ns:

```
var ns = ns.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the node sets in the model.

Arguments

- **Model** (Model)

Model that all node sets will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the node sets in model m, from 1000000:

```
NodeSet.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged node sets in the model.

Arguments

- **Model** (Model)

Model that all the flagged node sets will be renumbered in

- **flag** (Flag)

Flag set on the node sets that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the node sets in model m flagged with f, from 1000000:

```
NodeSet.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select node sets using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting node sets

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only node sets from that model can be selected. If the argument is a Flag then only node sets that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any node sets can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of node sets selected or null if menu cancelled

Return type

Number

Example

To select node sets from model m, flagging those selected with flag f, giving the prompt 'Select node sets':

```
NodeSet.Select(f, 'Select node sets', m);
```

To select node sets, flagging those selected with flag f but limiting selection to node sets flagged with flag l, giving the prompt 'Select node sets':

```
NodeSet.Select(f, 'Select node sets', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the node set.

Arguments

- **flag** (Flag)

Flag to set on the node set

Returns

No return value

Example

To set flag f for node set ns:

```
ns.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the node set. The node set will be sketched until you either call NodeSet.Unsketch(), NodeSet.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the node set is sketched. If omitted redraw is true. If you want to sketch several node sets and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch node set ns:

```
ns.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged node sets in the model. The node sets will be sketched until you either call NodeSet.Unsketch(), NodeSet.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged node sets will be sketched in

- **flag** (Flag)

Flag set on the node sets that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the node sets are sketched. If omitted redraw is true. If you want to sketch flagged node sets several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all node sets flagged with flag in model m:

```
NodeSet.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of node sets in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing node sets should be counted. If false or omitted referenced but undefined node sets will also be included in the total.

Returns

number of node sets

Return type

Number

Example

To get the total number of node sets in model m:

```
var total = NodeSet.Total(m);
```

Unblank()

Description

Unblanks the node set

Arguments

No arguments

Returns

No return value

Example

To unblank node set ns:

```
ns.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the node sets in the model.

Arguments

- **Model** (Model)

Model that all node sets will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the node sets in model m:

```
NodeSet.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged node sets in the model.

Arguments

- **Model** (Model)

Model that the flagged node sets will be unblanked in

- **flag** (Flag)

Flag set on the node sets that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the node sets in model m flagged with f:

```
NodeSet.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the node sets in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all node sets will be unset in

- **flag** (Flag)

Flag to unset on the node sets

Returns

No return value

Example

To unset the flag f on all the node sets in model m:

```
NodeSet.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the node set.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the node set is unsketched. If omitted redraw is true. If you want to unsketch several node sets and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch node set ns:

```
ns.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all node sets.

Arguments

- **Model** (Model)

Model that all node sets will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the node sets are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all node sets in model m:

```
NodeSet.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged node sets in the model.

Arguments

- **Model** (Model)

Model that all node sets will be unsketched in

- **flag** (Flag)

Flag set on the node sets that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the node sets are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all node sets flagged with flag in model m:

```
NodeSet.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

NodeSet object.

Return type

NodeSet

Example

To check if NodeSet property ns.example is a parameter by using the NodeSet.GetParameter() method:

```
if (ns.ViewParameters().GetParameter(ns.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for node set with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The node set object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for node set ns:

```
ns.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for node set ns:

```
ns.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for node set. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for node set ns:

```
ns.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this node set.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for node set ns:

```
var xrefs = ns.Xrefs();
```

---

toString()

Description

Creates a string containing the node_set data in keyword format. Note that this contains the keyword header and the keyword cards. See also NodeSet.Keyword() and NodeSet.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for node set n in keyword format

```
var s = n.toString();
```

RigidBodies class

The RigidBodies class gives you access to constrained rigid bodies cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- ExtractColour()
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

RigidBodies constants

Name	Description
RigidBodies.PART	RigidBodies is *CONSTRAINED_RIGID_BODIES.
RigidBodies.SET	RigidBodies is *CONSTRAINED_RIGID_BODIES_SET.

RigidBodies properties

Name	Type	Description
colour	Colour	The colour of the rigid body
exists (read only)	logical	true if rigid body merge exists, false if referred to but not defined.
iflag	integer	Flag for adding coonstrained mass properties to part inertia.
include	integer	The Include file number that the rigid body merge is in.
label (read only)	integer	The label the constrained rigid bodies has in PRIMER
model (read only)	integer	The Model number that the rigid body merge is in.
option	constant	The Constrained Rigid Bodies option. Can be RigidBodies.PART or RigidBodies.SET.
pidc	integer	Constrained rigid body part ID.
pidl	integer	Lead rigid body part ID.

Detailed Description

The RigidBodies class allows you to create, modify, edit and manipulate constrained rigid bodies cards. See the documentation below for more details.

Constructor

new RigidBodies(Model*[Model]*, options *[object]*)

Description

Create a new RigidBodies object.

Arguments

- **Model** (Model)

Model that constrained rigid bodies will be created in

- **options** (object)

Options specifying which properties would be used to create the keyword. If optional values are not used, then the default values below will be used.

Object has the following properties:

Name	Type	Description
iflag (optional)	integer	Flag for adding constrained mass properties to part inertia. (Default value 0)
option (optional)	constant	Specify the type of constrained rigid bodies. Can be RigidBodies.PART(default) or RigidBodies.SET
pidc	integer	Constrained rigid body part ID.
pidl	integer	Lead rigid body part ID.

Returns

RigidBodies object

Return type

RigidBodies

Example

To create a new constrained rigid bodies in model m with lead part 6 and constrained SET_PART 8

```
var output_obj    = new Object();
output_obj.pidl   = 6;
output_obj.pidc   = 8;
output_obj.iflag  = 1;
output_obj.option = RigidBodies.SET;

var cnst = new RigidBodies(m, output_obj);
```

Details of functions

AssociateComment(Comment[*Comment*])

Description

Associates a comment with a rigid body merge.

Arguments

- **Comment** (Comment)

Comment that will be attached to the rigid body merge

Returns

No return value

Example

To associate comment c to the rigid body merge m:

```
m.AssociateComment(c);
```

Blank()

Description

Blanks the rigid body merge

Arguments

No arguments

Returns

No return value

Example

To blank rigid body merge m:

```
m.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the rigid body merges in the model.

Arguments

- **Model** (Model)

Model that all rigid body merges will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the rigid body merges in model m:

```
RigidBodies.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged rigid body merges in the model.

Arguments

- **Model** (Model)

Model that all the flagged rigid body merges will be blanked in

- **flag** (Flag)

Flag set on the rigid body merges that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the rigid body merges in model m flagged with f:

```
RigidBodies.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the rigid body merge is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if rigid body merge m is blanked:

```
if (m.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse rigid body merge m:

```
m.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the rigid body merge.

Arguments

- **flag** (Flag)

Flag to clear on the rigid body merge

Returns

No return value

Example

To clear flag f for rigid body merge m:

```
m.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the rigid body merge. The target include of the copied rigid body merge can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

RigidBodies object

Return type

RigidBodies

Example

To copy rigid body merge m into rigid body merge z:

```
var z = m.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a rigid body merge

Arguments

- **Model** (Model)

Model that the rigid body merge will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

RigidBodies object (or null if not made).

Return type

RigidBodies

Example

To start creating a rigid body merge in model m:

```
var m = RigidBodies.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a rigid body merge.

Arguments

- **Comment** (Comment)

Comment that will be detached from the rigid body merge

Returns

No return value

Example

To detach comment c from the rigid body merge m:

```
m.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit rigid body merge m:

```
m.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for rigid body merge with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The rigid body merge object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for rigid body merge m:

```
m.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for rigid body merge m:

```
m.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for rigid body merge. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for rigid body merge m:

```
m.Error("My custom error");
```

ExtractColour()

Description

Extracts the **actual** colour used for rigid body merge.
By default in PRIMER many entities such as elements get their colour automatically from the part that they are in. PRIMER cycles through 13 default colours based on the label of the entity. In this case the rigid body merge colour property will return the value Colour.PART instead of the actual colour. This method will return the actual colour which is used for drawing the rigid body merge.

Arguments

No arguments

Returns

colour value (integer)

Return type

Number

Example

To return the colour used for drawing rigid body merge m:

```
var colour = m.ExtractColour();
```

First(Model[*Model*]) [static]

Description

Returns the first rigid body merge in the model.

Arguments

- **Model** (Model)

Model to get first rigid body merge in

Returns

RigidBodies object (or null if there are no rigid body merges in the model).

Return type

RigidBodies

Example

To get the first rigid body merge in model m:

```
var m = RigidBodies.First(m);
```

FlagAll(Model[*Model*], flag[*Flag*]) [static]

Description

Flags all of the rigid body merges in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all rigid body merges will be flagged in

- **flag** (Flag)

Flag to set on the rigid body merges

Returns

No return value

Example

To flag all of the rigid body merges with flag f in model m:

```
RigidBodies.FlagAll(m, f);
```

Flagged(flag[*Flag*])

Description

Checks if the rigid body merge is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the rigid body merge

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if rigid body merge m has flag f set on it:

```
if (m.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each rigid body merge in the model.
Note that ForEach has been designed to make looping over rigid body merges as fast as possible and so has some limitations.
Firstly, a single temporary RigidBodies object is created and on each function call it is updated with the current rigid body merge data. This means that you should not try to store the RigidBodies object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new rigid body merges inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all rigid body merges are in

- **func** (function)

Function to call for each rigid body merge

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the rigid body merges in model m:

```
RigidBodies.ForEach(m, test);
function test(m)
{
// m is RigidBodies object
}
```

To call function test for all of the rigid body merges in model m with optional object:

```
var data = { x:0, y:0 };
RigidBodies.ForEach(m, test, data);
function test(m, extra)
{
// m is RigidBodies object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of RigidBodies objects or properties for all of the rigid body merges in a model in PRIMER. If the optional property argument is not given then an array of RigidBodies objects is returned. If the property argument is given, that property value for each rigid body merge is returned in the array instead of a RigidBodies object

Arguments

- **Model** (Model)

Model to get rigid body merges from

- **property (optional)** (string)

Name for property to get for all rigid body merges in the model

Returns

Array of RigidBodies objects or properties

Return type

Array

Example

To make an array of RigidBodies objects for all of the rigid body merges in model m:

```
var a = RigidBodies.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each rigid body merge in model m:

```
var a = RigidBodies.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a rigid body merge.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the rigid body merge m:

```
var comm_array = m.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of RigidBodies objects for all of the flagged rigid body merges in a model in PRIMER If the optional property argument is not given then an array of RigidBodies objects is returned. If the property argument is given, then that property value for each rigid body merge is returned in the array instead of a RigidBodies object

Arguments

- **Model** (Model)

Model to get rigid body merges from

- **flag** (Flag)

Flag set on the rigid body merges that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged rigid body merges in the model

Returns

Array of RigidBodies objects or properties

Return type

Array

Example

To make an array of RigidBodies objects for all of the rigid body merges in model m flagged with f:

```
var m = RigidBodies.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the rigid body merges in model m flagged with f:

```
var a = RigidBodies.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the RigidBodies object for a rigid body merge ID.

Arguments

- **Model** (Model)

Model to find the rigid body merge in

- **number** (integer)

number of the rigid body merge you want the RigidBodies object for

Returns

RigidBodies object (or null if rigid body merge does not exist).

Return type

RigidBodies

Example

To get the RigidBodies object for rigid body merge 100 in model m

```
var m = RigidBodies.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a RigidBodies property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the RigidBodies.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

rigid body merge property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if RigidBodies property m.example is a parameter:

```
Options.property_parameter_names = true;
if (m.GetParameter(m.example) ) do_something...
Options.property_parameter_names = false;
```

To check if RigidBodies property m.example is a parameter by using the GetParameter method:

```
if (m.ViewParameters().GetParameter(m.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this constrained rigid bodies (*CONSTRAINED_RIGID_BODIES). **Note that a carriage return is not added**. See also RigidBodies.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for constrained rigid bodies r:

```
var key = r.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the constrained rigid bodies. **Note that a carriage return is not added**. See also RigidBodies.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for constrained rigid bodies r:

```
var cards = r.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last rigid body merge in the model.

Arguments

- **Model** (Model)

Model to get last rigid body merge in

Returns

RigidBodies object (or null if there are no rigid body merges in the model).

Return type

RigidBodies

Example

To get the last rigid body merge in model m:

```
var m = RigidBodies.Last(m);
```

Next()

Description

Returns the next rigid body merge in the model.

Arguments

No arguments

Returns

RigidBodies object (or null if there are no more rigid body merges in the model).

Return type

RigidBodies

Example

To get the rigid body merge in model m after rigid body merge m:

```
var m = m.Next();
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a rigid body merge.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only rigid body merges from that model can be picked. If the argument is a Flag then only rigid body merges that are flagged with *limit* can be selected. If omitted, or null, any rigid body merges from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

RigidBodies object (or null if not picked)

Return type

RigidBodies

Example

To pick a rigid body merge from model m giving the prompt 'Pick rigid body merge from screen':

```
var m = RigidBodies.Pick('Pick rigid body merge from screen', m);
```

Previous()

Description

Returns the previous rigid body merge in the model.

Arguments

No arguments

Returns

RigidBodies object (or null if there are no more rigid body merges in the model).

Return type

RigidBodies

Example

To get the rigid body merge in model m before rigid body merge m:

```
var m = m.Previous();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select rigid body merges using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting rigid body merges

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only rigid body merges from that model can be selected. If the argument is a Flag then only rigid body merges that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any rigid body merges can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of rigid body merges selected or null if menu cancelled

Return type

Number

Example

To select rigid body merges from model m, flagging those selected with flag f, giving the prompt ’Select rigid body merges’:

```
RigidBodies.Select(f, ’Select rigid body merges’, m);
```

To select rigid body merges, flagging those selected with flag f but limiting selection to rigid body merges flagged with flag l, giving the prompt ’Select rigid body merges’:

```
RigidBodies.Select(f, ’Select rigid body merges’, l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the rigid body merge.

Arguments

- **flag** (Flag)

Flag to set on the rigid body merge

Returns

No return value

Example

To set flag f for rigid body merge m:

```
m.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the rigid body merge. The rigid body merge will be sketched until you either call RigidBodies.Unsketch(), RigidBodies.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the rigid body merge is sketched. If omitted redraw is true. If you want to sketch several rigid body merges and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch rigid body merge m:

```
m.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged rigid body merges in the model. The rigid body merges will be sketched until you either call RigidBodies.Unsketch(), RigidBodies.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged rigid body merges will be sketched in

- **flag** (Flag)

Flag set on the rigid body merges that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the rigid body merges are sketched. If omitted redraw is true. If you want to sketch flagged rigid body merges several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all rigid body merges flagged with flag in model m:

```
RigidBodies.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of rigid body merges in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing rigid body merges should be counted. If false or omitted referenced but undefined rigid body merges will also be included in the total.

Returns

number of rigid body merges

Return type

Number

Example

To get the total number of rigid body merges in model m:

```
var total = RigidBodies.Total(m);
```

Unblank()

Description

Unblanks the rigid body merge

Arguments

No arguments

Returns

No return value

Example

To unblank rigid body merge m:

```
m.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the rigid body merges in the model.

Arguments

- **Model** (Model)

Model that all rigid body merges will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the rigid body merges in model m:

```
RigidBodies.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged rigid body merges in the model.

Arguments

- **Model** (Model)

Model that the flagged rigid body merges will be unblanked in

- **flag** (Flag)

Flag set on the rigid body merges that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the rigid body merges in model m flagged with f:

```
RigidBodies.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the rigid body merges in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all rigid body merges will be unset in

- **flag** (Flag)

Flag to unset on the rigid body merges

Returns

No return value

Example

To unset the flag f on all the rigid body merges in model m:

```
RigidBodies.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the rigid body merge.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the rigid body merge is unsketched. If omitted redraw is true. If you want to unsketch several rigid body merges and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch rigid body merge m:

```
m.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all rigid body merges.

Arguments

- **Model** (Model)

Model that all rigid body merges will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the rigid body merges are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all rigid body merges in model m:

```
RigidBodies.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged rigid body merges in the model.

Arguments

- **Model** (Model)

Model that all rigid body merges will be unsketched in

- **flag** (Flag)

Flag set on the rigid body merges that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the rigid body merges are unsketched. If omitted redraw is true. If you want to

unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all rigid body merges flagged with flag in model m:

```
RigidBodies.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

RigidBodies object.

Return type

RigidBodies

Example

To check if RigidBodies property m.example is a parameter by using the RigidBodies.GetParameter() method:

```
if (m.ViewParameters().GetParameter(m.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for rigid body merge with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The rigid body merge object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...

autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for rigid body merge m:

```
m.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for rigid body merge m:

```
m.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for rigid body merge. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for rigid body merge m:

```
m.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this rigid body merge.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for rigid body merge m:

```
var xrefs = m.Xrefs();
```

---

toString()

Description

Creates a string containing the constrained rigid bodies data in keyword format. Note that this contains the keyword header and the keyword cards. See also RigidBodies.Keyword() and RigidBodies.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for constrained rigid bodies r in keyword format

```
var s = r.toString();
```

Spotweld class

The Spotweld class gives you access to constrained spotweld cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- ExtractColour()
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)

- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

Spotweld properties

Name	Type	Description
colour	Colour	The colour of the spotweld
ep	real	Effective plastic strain at failure
exists (read only)	logical	true if spotweld exists, false if referred to but not defined.
filtered_force	logical	true if _FILTERED_FORCE option is set, false if not
id	logical	true if _ID option is set, false if not
include	integer	The Include file number that the spotweld is in.
label	integer	Constrained spotweld number
m	real	Exponent for shear spotweld force
model (read only)	integer	The Model number that the spotweld is in.
n	real	Exponent for normal spotweld force
n1	integer	Node ID
n2	integer	Node ID
nf	integer	Number of force vectors stored for filtering
sn	real	Normal force at spotweld failure
ss	real	Shear force at spotweld failure
tf	real	Failure time for nodal constraint set
tw	real	Time window for filtering
wid	integer	Constrained spotweld number (identical to label)

Detailed Description

The Spotweld class allows you to create, modify, edit and manipulate constrained spotweld cards. See the documentation below for more details.

Constructor

new Spotweld(Model*[Model]*, n1*[integer]*, n2*[integer]*, label (optional)*[integer]*)

Description

Create a new Spotweld object.

Arguments

- **Model** (Model)

Model that constrained spotweld will be created in

- **n1** (integer)

Node ID 1

- **n2** (integer)

Node ID 2

- **label (optional)** (integer)

Constrained spotweld number

Returns

Spotweld object

Return type

Spotweld

Example

To create a new constrained spotweld 500 in model m between nodes 10 and 11

```
var s = new Spotweld(m, 10, 11, 500);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a spotweld.

Arguments

- **Comment** (Comment)

Comment that will be attached to the spotweld

Returns

No return value

Example

To associate comment c to the spotweld s:

```
s.AssociateComment(c);
```

Blank()

Description

Blanks the spotweld

Arguments

No arguments

Returns

No return value

Example

To blank spotweld s:

```
s.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the spotwelds in the model.

Arguments

- **Model** (Model)

Model that all spotwelds will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the spotwelds in model m:

```
Spotweld.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged spotwelds in the model.

Arguments

- **Model** (Model)

Model that all the flagged spotwelds will be blanked in

- **flag** (Flag)

Flag set on the spotwelds that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the spotwelds in model m flagged with f:

```
Spotweld.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the spotweld is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if spotweld s is blanked:

```
if (s.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse spotweld s:

```
s.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the spotweld.

Arguments

- **flag** (Flag)

Flag to clear on the spotweld

Returns

No return value

Example

To clear flag f for spotweld s:

```
s.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the spotweld. The target include of the copied spotweld can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

Spotweld object

Return type

Spotweld

Example

To copy spotweld s into spotweld z:

```
var z = s.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a spotweld

Arguments

- **Model** (Model)

Model that the spotweld will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

Spotweld object (or null if not made).

Return type

Spotweld

Example

To start creating a spotweld in model m:

```
var s = Spotweld.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a spotweld.

Arguments

- **Comment** (Comment)

Comment that will be detached from the spotweld

Returns

No return value

Example

To detach comment c from the spotweld s:

```
s.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit spotweld s:

```
s.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for spotweld with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The spotweld object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for spotweld s:

```
s.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for spotweld s:

```
s.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for spotweld. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for spotweld s:

```
s.Error("My custom error");
```

ExtractColour()

Description

Extracts the **actual** colour used for spotweld.
By default in PRIMER many entities such as elements get their colour automatically from the part that they are in. PRIMER cycles through 13 default colours based on the label of the entity. In this case the spotweld colour property will return the value Colour.PART instead of the actual colour. This method will return the actual colour which is used for drawing the spotweld.

Arguments

No arguments

Returns

colour value (integer)

Return type

Number

Example

To return the colour used for drawing spotweld s:

```
var colour = s.ExtractColour();
```

First(Model*[Model]*) [static]

Description

Returns the first spotweld in the model.

Arguments

- **Model** (Model)

Model to get first spotweld in

Returns

Spotweld object (or null if there are no spotwelds in the model).

Return type

Spotweld

Example

To get the first spotweld in model m:

```
var s = Spotweld.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free spotweld label in the model. Also see Spotweld.LastFreeLabel(), Spotweld.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free spotweld label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

Spotweld label.

Return type

Number

Example

To get the first free spotweld label in model m:

```
var label = Spotweld.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the spotwelds in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all spotwelds will be flagged in

- **flag** (Flag)

Flag to set on the spotwelds

Returns

No return value

Example

To flag all of the spotwelds with flag f in model m:

```
Spotweld.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the spotweld is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the spotweld

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if spotweld s has flag f set on it:

```
if (s.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each spotweld in the model.
Note that ForEach has been designed to make looping over spotwelds as fast as possible and so has some limitations.
Firstly, a single temporary Spotweld object is created and on each function call it is updated with the current spotweld data. This means that you should not try to store the Spotweld object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new spotwelds inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all spotwelds are in

- **func** (function)

Function to call for each spotweld

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the spotwelds in model m:

```
Spotweld.ForEach(m, test);
function test(s)
{
// s is Spotweld object
}
```

To call function test for all of the spotwelds in model m with optional object:

```
var data = { x:0, y:0 };
Spotweld.ForEach(m, test, data);
function test(s, extra)
{
// s is Spotweld object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Spotweld objects or properties for all of the spotwelds in a model in PRIMER. If the optional property argument is not given then an array of Spotweld objects is returned. If the property argument is given, that property value for each spotweld is returned in the array instead of a Spotweld object

Arguments

- **Model** (Model)

Model to get spotwelds from

- **property (optional)** (string)

Name for property to get for all spotwelds in the model

Returns

Array of Spotweld objects or properties

Return type

Array

Example

To make an array of Spotweld objects for all of the spotwelds in model m:

```
var a = Spotweld.GetAll(m);
```

To return an array containing the value of property ’foo’ (for example ’x’ for a node) for each spotweld in model m:

```
var a = Spotweld.GetAll(m, ’foo’);
```

GetComments()

Description

Extracts the comments associated to a spotweld.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the spotweld s:

```
var comm_array = s.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of Spotweld objects for all of the flagged spotwelds in a model in PRIMER If the optional property argument is not given then an array of Spotweld objects is returned. If the property argument is given, then that property value for each spotweld is returned in the array instead of a Spotweld object

Arguments

- **Model** (Model)

Model to get spotwelds from

- **flag** (Flag)

Flag set on the spotwelds that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged spotwelds in the model

Returns

Array of Spotweld objects or properties

Return type

Array

Example

To make an array of Spotweld objects for all of the spotwelds in model m flagged with f:

```
var s = Spotweld.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the spotwelds in model m flagged with f:

```
var a = Spotweld.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the Spotweld object for a spotweld ID.

Arguments

- **Model** (Model)

Model to find the spotweld in

- **number** (integer)

number of the spotweld you want the Spotweld object for

Returns

Spotweld object (or null if spotweld does not exist).

Return type

Spotweld

Example

To get the Spotweld object for spotweld 100 in model m

```
var s = Spotweld.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a Spotweld property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the Spotweld.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

spotweld property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if Spotweld property s.example is a parameter:

```
Options.property_parameter_names = true;
if (s.GetParameter(s.example) ) do_something...
Options.property_parameter_names = false;
```

To check if Spotweld property s.example is a parameter by using the GetParameter method:

```
if (s.ViewParameters().GetParameter(s.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this spotweld (*CONSTRAINED_SPOTWELD). **Note that a carriage return is not added**. See also Spotweld.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for spotweld s:

```
var key = s.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the spotweld. **Note that a carriage return is not added**. See also Spotweld.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for spotweld s:

```
var cards = s.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last spotweld in the model.

Arguments

- **Model** (Model)

Model to get last spotweld in

Returns

Spotweld object (or null if there are no spotwelds in the model).

Return type

Spotweld

Example

To get the last spotweld in model m:

```
var s = Spotweld.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free spotweld label in the model. Also see Spotweld.FirstFreeLabel(), Spotweld.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free spotweld label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

Spotweld label.

Return type

Number

Example

To get the last free spotweld label in model m:

```
var label = Spotweld.LastFreeLabel(m);
```

Next()

Description

Returns the next spotweld in the model.

Arguments

No arguments

Returns

Spotweld object (or null if there are no more spotwelds in the model).

Return type

Spotweld

Example

To get the spotweld in model m after spotweld s:

```
var s = s.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) spotweld label in the model. Also see Spotweld.FirstFreeLabel(), Spotweld.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free spotweld label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

Spotweld label.

Return type

Number

Example

To get the next free spotweld label in model m:

```
var label = Spotweld.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a spotweld.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only spotwelds from that model can be picked. If the argument is a Flag then only spotwelds that are flagged with *limit* can be selected. If omitted, or null, any spotwelds from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

Spotweld object (or null if not picked)

Return type

Spotweld

Example

To pick a spotweld from model m giving the prompt 'Pick spotweld from screen':

```
var s = Spotweld.Pick('Pick spotweld from screen', m);
```

Previous()

Description

Returns the previous spotweld in the model.

Arguments

No arguments

Returns

Spotweld object (or null if there are no more spotwelds in the model).

Return type

Spotweld

Example

To get the spotweld in model m before spotweld s:

```
var s = s.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the spotwelds in the model.

Arguments

- **Model** (Model)

Model that all spotwelds will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the spotwelds in model m, from 1000000:

```
Spotweld.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged spotwelds in the model.

Arguments

- **Model** (Model)

Model that all the flagged spotwelds will be renumbered in

- **flag** (Flag)

Flag set on the spotwelds that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the spotwelds in model m flagged with f, from 1000000:

```
Spotweld.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select spotwelds using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting spotwelds

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only spotwelds from that model can be selected. If the argument is a Flag then only spotwelds that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any spotwelds can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of spotwelds selected or null if menu cancelled

Return type

Number

Example

To select spotwelds from model m, flagging those selected with flag f, giving the prompt 'Select spotwelds':

```
Spotweld.Select(f, 'Select spotwelds', m);
```

To select spotwelds, flagging those selected with flag f but limiting selection to spotwelds flagged with flag l, giving the prompt 'Select spotwelds':

```
Spotweld.Select(f, 'Select spotwelds', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the spotweld.

Arguments

- **flag** (Flag)

Flag to set on the spotweld

Returns

No return value

Example

To set flag f for spotweld s:

```
s.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the spotweld. The spotweld will be sketched until you either call Spotweld.Unsketch(), Spotweld.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the spotweld is sketched. If omitted redraw is true. If you want to sketch several spotwelds and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch spotweld s:

```
s.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged spotwelds in the model. The spotwelds will be sketched until you either call Spotweld.Unsketch(), Spotweld.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged spotwelds will be sketched in

- **flag** (Flag)

Flag set on the spotwelds that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the spotwelds are sketched. If omitted redraw is true. If you want to sketch flagged spotwelds several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all spotwelds flagged with flag in model m:

```
Spotweld.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of spotwelds in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing spotwelds should be counted. If false or omitted referenced but undefined spotwelds will also be included in the total.

Returns

number of spotwelds

Return type

Number

Example

To get the total number of spotwelds in model m:

```
var total = Spotweld.Total(m);
```

Unblank()

Description

Unblanks the spotweld

Arguments

No arguments

Returns

No return value

Example

To unblank spotweld s:

```
s.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the spotwelds in the model.

Arguments

- **Model** (Model)

Model that all spotwelds will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the spotwelds in model m:

```
Spotweld.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged spotwelds in the model.

Arguments

- **Model** (Model)

Model that the flagged spotwelds will be unblanked in

- **flag** (Flag)

Flag set on the spotwelds that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the spotwelds in model m flagged with f:

```
Spotweld.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the spotwelds in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all spotwelds will be unset in

- **flag** (Flag)

Flag to unset on the spotwelds

Returns

No return value

Example

To unset the flag f on all the spotwelds in model m:

```
Spotweld.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the spotweld.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the spotweld is unsketched. If omitted redraw is true. If you want to unsketch several spotwelds and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch spotweld s:

```
s.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all spotwelds.

Arguments

- **Model** (Model)

Model that all spotwelds will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the spotwelds are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all spotwelds in model m:

```
Spotweld.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged spotwelds in the model.

Arguments

- **Model** (Model)

Model that all spotwelds will be unsketched in

- **flag** (Flag)

Flag set on the spotwelds that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the spotwelds are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all spotwelds flagged with flag in model m:

```
Spotweld.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

Spotweld object.

Return type

Spotweld

Example

To check if Spotweld property s.example is a parameter by using the Spotweld.GetParameter() method:

```
if (s.ViewParameters().GetParameter(s.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for spotweld with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The spotweld object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for spotweld s:

```
s.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for spotweld s:

```
s.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for spotweld. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for spotweld s:

```
s.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this spotweld.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for spotweld s:

```
var xrefs = s.Xrefs();
```

---

toString()

Description

Creates a string containing the spotweld data in keyword format. Note that this contains the keyword header and the keyword cards. See also Spotweld.Keyword() and Spotweld.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for spotweld s in keyword format

```
var str = s.toString();
```

Spr2 class

The Spr2 class gives you access to constrained spr2 cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- First(Model*[Model]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

Spr2 constants

Constants for Flags for Interpolation

Name	Description
Spr2.INVERSE	Property INTP value EQ.2.0: Inverse distance weighting.
Spr2.LINEAR	Property INTP value EQ.0.0: Linear (default).
Spr2.UNIFORM	Property INTP value EQ.1.0: Uniform.

Spr2 properties

Name	Type	Description
alpha1	real	Dimensionless parameter scaling the effective displacement.
alpha2	real	Dimensionless parameter scaling the effective displacement.
alpha3	real	Dimensionless parameter scaling the effective displacement. (GT.0: incremental update (default), LT.0: total update (recommended))
d	real	Rivet diameter.
dens	real	Rivet density (necessary for time step calculation).
dn	real	Failure displacement in normal direction.
dt	real	Failure displacement in tangential direction.
exists (read only)	logical	true if spr2 exists, false if referred to but not defined.
expn	real	Exponent value for load function in normal direction.
expt	real	Exponent value for load function in tangential direction.
fn	real	Rivet strength in tension (pull-out) or (if -ve) label for UPID
ft	real	Rivet strength in pure shear.
include	integer	The Include file number that the spr2 is in.
intp	real	Flag for interpolation. Values can be Spr2.LINEAR, Spr2.UNIFORM or Spr2.INVERSE.
lpid	integer	Lower Sheet Part ID.
model (read only)	integer	The Model number that the spr2 is in.
nsid	integer	Node Set ID of rivet location nodes.
pidvb	integer	Part ID for visualization beams representing SPR2 in postprocessing.
thick	real	Total thickness of lead and constrained sheet.
upid	integer	Upper Sheet Part ID.
xin	real	Fraction of failure displacement at maximum normal force.
xit	real	Fraction of failure displacement at maximum tangential force.
xpid1	integer	Extra Part ID 1 for multi-sheet connection.
xpid2	integer	Extra Part ID 2 for multi-sheet connection.
xpid3	integer	Extra Part ID 3 for multi-sheet connection.
xpid4	integer	Extra Part ID 4 for multi-sheet connection.

Detailed Description

The Spr2 class allows you to create, modify, edit and manipulate constrained spr2 cards. See the documentation below for more details.

Constructor

new Spr2(Model*[Model]*, upid*[integer]*, lpid*[integer]*, nsid*[integer]*)

Description

Create a new Spr2 object.

Arguments

- **Model** (Model)

Model that constrained spr2 will be created in

- **upid** (integer)

Upper Sheet Part ID.

- **lpid** (integer)

Lower Sheet Part ID

- **nsid** (integer)

Node Set ID of rivet location nodes.

Returns

Spr2 object

Return type

Spr2

Example

To create a new constrained spr2 in model m with lead sheet 100, constrained sheet 200 and rivet node set 100

```
var s = new Spr2(m, 100, 200, 100);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a spr2.

Arguments

- **Comment** (Comment)

Comment that will be attached to the spr2

Returns

No return value

Example

To associate comment c to the spr2 s:

```
s.AssociateComment(c);
```

Blank()

Description

Blanks the spr2

Arguments

No arguments

Returns

No return value

Example

To blank spr2 s:

```
s.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the spr2s in the model.

Arguments

- **Model** (Model)

Model that all spr2s will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the spr2s in model m:

```
Spr2.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged spr2s in the model.

Arguments

- **Model** (Model)

Model that all the flagged spr2s will be blanked in

- **flag** (Flag)

Flag set on the spr2s that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the spr2s in model m flagged with f:

```
Spr2.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the spr2 is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if spr2 s is blanked:

```
if (s.Blanked() ) do_something...
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the spr2.

Arguments

- **flag** (Flag)

Flag to clear on the spr2

Returns

No return value

Example

To clear flag f for spr2 s:

```
s.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the spr2. The target include of the copied spr2 can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

Spr2 object

Return type

Spr2

Example

To copy spr2 s into spr2 z:

```
var z = s.Copy();
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a spr2.

Arguments

- **Comment** (Comment)

Comment that will be detached from the spr2

Returns

No return value

Example

To detach comment c from the spr2 s:

```
s.DetachComment(c);
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for spr2 with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The spr2 object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...

autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for spr2 s:

```
s.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for spr2 s:

```
s.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for spr2. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for spr2 s:

```
s.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first spr2 in the model.

Arguments

- **Model** (Model)

Model to get first spr2 in

Returns

Spr2 object (or null if there are no spr2s in the model).

Return type

Spr2

Example

To get the first spr2 in model m:

```
var s = Spr2.First(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the spr2s in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all spr2s will be flagged in

- **flag** (Flag)

Flag to set on the spr2s

Returns

No return value

Example

To flag all of the spr2s with flag f in model m:

```
Spr2.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the spr2 is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the spr2

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if spr2 s has flag f set on it:

```
if (s.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each spr2 in the model.
Note that ForEach has been designed to make looping over spr2s as fast as possible and so has some limitations. Firstly, a single temporary Spr2 object is created and on each function call it is updated with the current spr2 data. This means that you should not try to store the Spr2 object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new spr2s inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all spr2s are in

- **func** (function)

Function to call for each spr2

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the spr2s in model m:

```
Spr2.ForEach(m, test);
function test(s)
{
// s is Spr2 object
}
```

To call function test for all of the spr2s in model m with optional object:

```
var data = { x:0, y:0 };
Spr2.ForEach(m, test, data);
function test(s, extra)
{
// s is Spr2 object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Spr2 objects or properties for all of the spr2s in a model in PRIMER. If the optional property argument is not given then an array of Spr2 objects is returned. If the property argument is given, that property value for each spr2 is returned in the array instead of a Spr2 object

Arguments

- **Model** (Model)

Model to get spr2s from

- **property (optional)** (string)

Name for property to get for all spr2s in the model

Returns

Array of Spr2 objects or properties

Return type

Array

Example

To make an array of Spr2 objects for all of the spr2s in model m:

```
var a = Spr2.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each spr2 in model m:

```
var a = Spr2.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a spr2.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the spr2 s:

```
var comm_array = s.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of Spr2 objects for all of the flagged spr2s in a model in PRIMER If the optional property argument is not given then an array of Spr2 objects is returned. If the property argument is given, then that property value for each spr2 is returned in the array instead of a Spr2 object

Arguments

- **Model** (Model)

Model to get spr2s from

- **flag** (Flag)

Flag set on the spr2s that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged spr2s in the model

Returns

Array of Spr2 objects or properties

Return type

Array

Example

To make an array of Spr2 objects for all of the spr2s in model m flagged with f:

```
var s = Spr2.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the spr2s in model m flagged with f:

```
var a = Spr2.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the Spr2 object for a spr2 ID.

Arguments

- **Model** (Model)

Model to find the spr2 in

- **number** (integer)

number of the spr2 you want the Spr2 object for

Returns

Spr2 object (or null if spr2 does not exist).

Return type

Spr2

Example

To get the Spr2 object for spr2 100 in model m

```
var s = Spr2.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a Spr2 property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the Spr2.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

spr2 property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if Spr2 property s.example is a parameter:

```
Options.property_parameter_names = true;
if (s.GetParameter(s.example) ) do_something...
Options.property_parameter_names = false;
```

To check if Spr2 property s.example is a parameter by using the GetParameter method:

```
if (s.ViewParameters().GetParameter(s.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this spr2 (*CONSTRAINED_SPR2). **Note that a carriage return is not added**. See also Spr2.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for spr2 s:

```
var key = s.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the spr2. **Note that a carriage return is not added**. See also Spr2.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for spr2 s:

```
var cards = s.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last spr2 in the model.

Arguments

- **Model** (Model)

Model to get last spr2 in

Returns

Spr2 object (or null if there are no spr2s in the model).

Return type

Spr2

Example

To get the last spr2 in model m:

```
var s = Spr2.Last(m);
```

Next()

Description

Returns the next spr2 in the model.

Arguments

No arguments

Returns

Spr2 object (or null if there are no more spr2s in the model).

Return type

Spr2

Example

To get the spr2 in model m after spr2 s:

```
var s = s.Next();
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a spr2.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only spr2s from that model can be picked. If the argument is a Flag then only spr2s that are flagged with *limit* can be selected. If omitted, or null, any spr2s from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

Spr2 object (or null if not picked)

Return type

Spr2

Example

To pick a spr2 from model m giving the prompt 'Pick spr2 from screen':

```
var s = Spr2.Pick('Pick spr2 from screen', m);
```

Previous()

Description

Returns the previous spr2 in the model.

Arguments

No arguments

Returns

Spr2 object (or null if there are no more spr2s in the model).

Return type

Spr2

Example

To get the spr2 in model m before spr2 s:

```
var s = s.Previous();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select spr2s using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting spr2s

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only spr2s from that model can be selected. If the argument is a Flag then only spr2s that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any spr2s can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of spr2s selected or null if menu cancelled

Return type

Number

Example

To select spr2s from model m, flagging those selected with flag f, giving the prompt 'Select spr2s':

```
Spr2.Select(f, 'Select spr2s', m);
```

To select spr2s, flagging those selected with flag f but limiting selection to spr2s flagged with flag l, giving the prompt 'Select spr2s':

```
Spr2.Select(f, 'Select spr2s', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the spr2.

Arguments

- **flag** (Flag)

Flag to set on the spr2

Returns

No return value

Example

To set flag f for spr2 s:

```
s.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the spr2. The spr2 will be sketched until you either call Spr2.Unsketch(), Spr2.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the spr2 is sketched. If omitted redraw is true. If you want to sketch several spr2s and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch spr2 s:

```
s.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged spr2s in the model. The spr2s will be sketched until you either call Spr2.Unsketch(), Spr2.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged spr2s will be sketched in

- **flag** (Flag)

Flag set on the spr2s that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the spr2s are sketched. If omitted redraw is true. If you want to sketch flagged spr2s several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all spr2s flagged with flag in model m:

```
Spr2.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of spr2s in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing spr2s should be counted. If false or omitted referenced but undefined spr2s will also be included in the total.

Returns

number of spr2s

Return type

Number

Example

To get the total number of spr2s in model m:

```
var total = Spr2.Total(m);
```

Unblank()

Description

Unblanks the spr2

Arguments

No arguments

Returns

No return value

Example

To unblank spr2 s:

```
s.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the spr2s in the model.

Arguments

- **Model** (Model)

Model that all spr2s will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the spr2s in model m:

```
Spr2.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged spr2s in the model.

Arguments

- **Model** (Model)

Model that the flagged spr2s will be unblanked in

- **flag** (Flag)

Flag set on the spr2s that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the spr2s in model m flagged with f:

```
Spr2.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the spr2s in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all spr2s will be unset in

- **flag** (Flag)

Flag to unset on the spr2s

Returns

No return value

Example

To unset the flag f on all the spr2s in model m:

```
Spr2.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the spr2.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the spr2 is unsketched. If omitted redraw is true. If you want to unsketch several spr2s and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch spr2 s:

```
s.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all spr2s.

Arguments

- **Model** (Model)

Model that all spr2s will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the spr2s are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all spr2s in model m:

```
Spr2.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged spr2s in the model.

Arguments

- **Model** (Model)

Model that all spr2s will be unsketched in

- **flag** (Flag)

Flag set on the spr2s that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the spr2s are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all spr2s flagged with flag in model m:

```
Spr2.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

Spr2 object.

Return type

Spr2

Example

To check if Spr2 property s.example is a parameter by using the Spr2.GetParameter() method:

```
if (s.ViewParameters().GetParameter(s.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for spr2 with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The spr2 object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for spr2 s:

```
s.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for spr2 s:

```
s.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for spr2. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for spr2 s:

```
s.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this spr2.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for spr2 s:

```
var xrefs = s.Xrefs();
```

---

toString()

Description

Creates a string containing the spr2 data in keyword format. Note that this contains the keyword header and the keyword cards. See also Spr2.Keyword() and Spr2.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for spr2 s in keyword format

```
var str = s.toString();
```

TieBreak class

The TieBreak class gives you access to constrained Tie-Break cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- First(Model*[Model]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

TieBreak properties

Name	Type	Description

eppf	real	Plastic strain at failure.
exists (read only)	logical	true if tie-break exists, false if referred to but not defined.
include	integer	The Include file number that the tie-break is in.
model (read only)	integer	The Model number that the tie-break is in.
nsid1	integer	First Node Set ID.
nsid2	integer	Second Node Set ID.

Detailed Description

The TieBreak class allows you to create, modify, edit and manipulate constrained tie-break cards. See the documentation below for more details.

Constructor

new TieBreak(Model*[Model]*, nsid1*[integer]*, nsid2*[integer]*, eppf (optional)*[real]*)

Description

Create a new TieBreak object.

Arguments

- **Model** (Model)

Model that constrained tie-break will be created in

- **nsid1** (integer)

First Node Set ID.

- **nsid2** (integer)

Second Node Set ID.

- **eppf (optional)** (real)

Plastic strain at failure.

Returns

TieBreak object

Return type

TieBreak

Example

To create a new constrained tie-break in model m with first node set 100, second node set 200

```
var tb = new TieBreak(m, 100, 200);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a tie-break.

Arguments

- **Comment** (Comment)

Comment that will be attached to the tie-break

Returns

No return value

Example

To associate comment c to the tie-break tb:

```
tb.AssociateComment(c);
```

Blank()

Description

Blanks the tie-break

Arguments

No arguments

Returns

No return value

Example

To blank tie-break tb:

```
tb.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the tie-breaks in the model.

Arguments

- **Model** (Model)

Model that all tie-breaks will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the tie-breaks in model m:

```
TieBreak.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged tie-breaks in the model.

Arguments

- **Model** (Model)

Model that all the flagged tie-breaks will be blanked in

- **flag** (Flag)

Flag set on the tie-breaks that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the tie-breaks in model m flagged with f:

```
TieBreak.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the tie-break is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if tie-break tb is blanked:

```
if (tb.Blanked() ) do_something...
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the tie-break.

Arguments

- **flag** (Flag)

Flag to clear on the tie-break

Returns

No return value

Example

To clear flag f for tie-break tb:

```
tb.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the tie-break. The target include of the copied tie-break can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

TieBreak object

Return type

TieBreak

Example

To copy tie-break tb into tie-break z:

```
var z = tb.Copy();
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a tie-break.

Arguments

- **Comment** (Comment)

Comment that will be detached from the tie-break

Returns

No return value

Example

To detach comment c from the tie-break tb:

```
tb.DetachComment(c);
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for tie-break with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description

autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The tie-break object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for tie-break tb:

```
tb.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for tie-break tb:

```
tb.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for tie-break. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for tie-break tb:

```
tb.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first tie-break in the model.

Arguments

- **Model** (Model)

Model to get first tie-break in

Returns

TieBreak object (or null if there are no tie-breaks in the model).

Return type

TieBreak

Example

To get the first tie-break in model m:

```
var tb = TieBreak.First(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the tie-breaks in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all tie-breaks will be flagged in

- **flag** (Flag)

Flag to set on the tie-breaks

Returns

No return value

Example

To flag all of the tie-breaks with flag f in model m:

```
TieBreak.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the tie-break is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the tie-break

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if tie-break tb has flag f set on it:

```
if (tb.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each tie-break in the model.
Note that ForEach has been designed to make looping over tie-breaks as fast as possible and so has some limitations.
Firstly, a single temporary TieBreak object is created and on each function call it is updated with the current tie-break data. This means that you should not try to store the TieBreak object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new tie-breaks inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all tie-breaks are in

- **func** (function)

Function to call for each tie-break

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the tie-breaks in model m:

```
TieBreak.ForEach(m, test);
function test(tb)
{
// tb is TieBreak object
}
```

To call function test for all of the tie-breaks in model m with optional object:

```
var data = { x:0, y:0 };
TieBreak.ForEach(m, test, data);
function test(tb, extra)
{
// tb is TieBreak object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of TieBreak objects or properties for all of the tie-breaks in a model in PRIMER. If the optional property argument is not given then an array of TieBreak objects is returned. If the property argument is given, that property value for each tie-break is returned in the array instead of a TieBreak object

Arguments

- **Model** (Model)

Model to get tie-breaks from

- **property (optional)** (string)

Name for property to get for all tie-breaks in the model

Returns

Array of TieBreak objects or properties

Return type

Array

Example

To make an array of TieBreak objects for all of the tie-breaks in model m:

```
var a = TieBreak.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each tie-break in model m:

```
var a = TieBreak.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a tie-break.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the tie-break tb:

```
var comm_array = tb.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of TieBreak objects for all of the flagged tie-breaks in a model in PRIMER If the optional property argument is not given then an array of TieBreak objects is returned. If the property argument is given, then that property value for each tie-break is returned in the array instead of a TieBreak object

Arguments

- **Model** (Model)

Model to get tie-breaks from

- **flag** (Flag)

Flag set on the tie-breaks that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged tie-breaks in the model

Returns

Array of TieBreak objects or properties

Return type

Array

Example

To make an array of TieBreak objects for all of the tie-breaks in model m flagged with f:

```
var tb = TieBreak.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the tie-breaks in model m flagged with f:

```
var a = TieBreak.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the TieBreak object for a tie-break ID.

Arguments

- **Model** (Model)

Model to find the tie-break in

- **number** (integer)

number of the tie-break you want the TieBreak object for

Returns

TieBreak object (or null if tie-break does not exist).

Return type

TieBreak

Example

To get the TieBreak object for tie-break 100 in model m

```
var tb = TieBreak.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a TieBreak property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the TieBreak.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

tie-break property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if TieBreak property tb.example is a parameter:

```
Options.property_parameter_names = true;
if (tb.GetParameter(tb.example) ) do_something...
Options.property_parameter_names = false;
```

To check if TieBreak property tb.example is a parameter by using the GetParameter method:

```
if (tb.ViewParameters().GetParameter(tb.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this tie-break (**CONSTRAINED_TIE_BREAK). **Note that a carriage return is not added**. See also TieBreak.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for tie-break tb:

```
var key = tb.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the tie-break. **Note that a carriage return is not added**. See also TieBreak.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for tie-break tb:

```
var cards = tb.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last tie-break in the model.

Arguments

- **Model** (Model)

Model to get last tie-break in

Returns

TieBreak object (or null if there are no tie-breaks in the model).

Return type

TieBreak

Example

To get the last tie-break in model m:

```
var tb = TieBreak.Last(m);
```

Next()

Description

Returns the next tie-break in the model.

Arguments

No arguments

Returns

TieBreak object (or null if there are no more tie-breaks in the model).

Return type

TieBreak

Example

To get the tie-break in model m after tie-break tb:

```
var tb = tb.Next();
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a tie-break.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only tie-breaks from that model can be picked. If the argument is a Flag then only tie-breaks that are flagged with *limit* can be selected. If omitted, or null, any tie-breaks from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

TieBreak object (or null if not picked)

Return type

TieBreak

Example

To pick a tie-break from model m giving the prompt 'Pick tie-break from screen':

```
var tb = TieBreak.Pick('Pick tie-break from screen', m);
```

Previous()

Description

Returns the previous tie-break in the model.

Arguments

No arguments

Returns

TieBreak object (or null if there are no more tie-breaks in the model).

Return type

TieBreak

Example

To get the tie-break in model m before tie-break tb:

```
var tb = tb.Previous();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select tie-breaks using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting tie-breaks

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only tie-breaks from that model can be selected. If the argument is a Flag then only tie-breaks that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any tie-breaks can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of tie-breaks selected or null if menu cancelled

Return type

Number

Example

To select tie-breaks from model m, flagging those selected with flag f, giving the prompt 'Select tie-breaks':

```
TieBreak.Select(f, 'Select tie-breaks', m);
```

To select tie-breaks, flagging those selected with flag f but limiting selection to tie-breaks flagged with flag l, giving the prompt 'Select tie-breaks':

```
TieBreak.Select(f, 'Select tie-breaks', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the tie-break.

Arguments

- **flag** (Flag)

Flag to set on the tie-break

Returns

No return value

Example

To set flag f for tie-break tb:

```
tb.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the tie-break. The tie-break will be sketched until you either call TieBreak.Unsketch(), TieBreak.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the tie-break is sketched. If omitted redraw is true. If you want to sketch several tie-breaks and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch tie-break tb:

```
tb.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged tie-breaks in the model. The tie-breaks will be sketched until you either call TieBreak.Unsketch(), TieBreak.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged tie-breaks will be sketched in

- **flag** (Flag)

Flag set on the tie-breaks that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the tie-breaks are sketched. If omitted redraw is true. If you want to sketch flagged tie-breaks several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all tie-breaks flagged with flag in model m:

```
TieBreak.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of tie-breaks in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing tie-breaks should be counted. If false or omitted referenced but undefined tie-breaks will also be included in the total.

Returns

number of tie-breaks

Return type

Number

Example

To get the total number of tie-breaks in model m:

```
var total = TieBreak.Total(m);
```

Unblank()

Description

Unblanks the tie-break

Arguments

No arguments

Returns

No return value

Example

To unblank tie-break tb:

```
tb.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the tie-breaks in the model.

Arguments

- **Model** (Model)

Model that all tie-breaks will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the tie-breaks in model m:

```
TieBreak.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged tie-breaks in the model.

Arguments

- **Model** (Model)

Model that the flagged tie-breaks will be unblanked in

- **flag** (Flag)

Flag set on the tie-breaks that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the tie-breaks in model m flagged with f:

```
TieBreak.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the tie-breaks in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all tie-breaks will be unset in

- **flag** (Flag)

Flag to unset on the tie-breaks

Returns

No return value

Example

To unset the flag f on all the tie-breaks in model m:

```
TieBreak.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the tie-break.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the tie-break is unsketched. If omitted redraw is true. If you want to unsketch several tie-breaks and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch tie-break tb:

```
tb.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all tie-breaks.

Arguments

- **Model** (Model)

Model that all tie-breaks will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the tie-breaks are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all tie-breaks in model m:

```
TieBreak.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged tie-breaks in the model.

Arguments

- **Model** (Model)

Model that all tie-breaks will be unsketched in

- **flag** (Flag)

Flag set on the tie-breaks that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the tie-breaks are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all tie-breaks flagged with flag in model m:

```
TieBreak.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

TieBreak object.

Return type

TieBreak

Example

To check if TieBreak property tb.example is a parameter by using the TieBreak.GetParameter() method:

```
if (tb.ViewParameters().GetParameter(tb.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for tie-break with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The tie-break object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for tie-break tb:

```
tb.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for tie-break tb:

```
tb.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for tie-break. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for tie-break tb:

```
tb.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this tie-break.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for tie-break tb:

```
var xrefs = tb.Xrefs();
```

---

toString()

Description

Creates a string containing the tie-break data in keyword format. Note that this contains the keyword header and the keyword cards. See also TieBreak.Keyword() and TieBreak.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for tie-break tb in keyword format

```
var str = tb.toString();
```

Contact class

The Contact class gives you access to define contact cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- WriteToExcel(flag*[Flag]*, filename*[string]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Constrained(connection (optional)*[boolean]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- ExtractColour()
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Interactions(type (optional)*[constant]*)
- Keyword()
- KeywordCards()
- Next()
- PenCheck(flag*[Flag]*, eflag*[integer]*)
- PenCheckEdit(modal (optional)*[boolean]*, check_mode (optional)*[constant]*, mpp_threshold (optional)*[real]*, report_crossed_3d_elems (optional)*[boolean]*, contact_penchk_dup_shells (optional)*[constant]*)
- Previous()

- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- StatusCheck()
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

Contact constants

Name	Description
Contact.CROSSED_EDGES	Return crossed edges in Contact.Interactions()
Contact.MPP_MODE	MPP penetration check mode
Contact.PENETRATIONS	Return penetrations in Contact.Interactions()
Contact.SMP_MODE	SMP penetration check mode

Constants for Contact _OFFSET types

Name	Description
Contact.BEAM_OFFSET	Adds _BEAM_OFFSET option
Contact.CONSTR_OFFSET	Adds _CONSTRAINED_OFFSET option
Contact.NO_OFFSET	No offset option added.
Contact.SIMPLE_OFFSET	Adds _OFFSET option

Constants for Contact penetration check_mode types

Name	Description
Contact.MPP_METHOD	Launches the penetration edit panel with the MPP methodology turned on
Contact.SMP_METHOD	Launches the penetration edit panel with the SMP methodology turned on

Constants for Contact penetration contact_penchk_dup_shells types

Name	Description
Contact.SHELL_AUTO	Launches the penetration edit panel with Automatic shell treatment of duplicate shells.
Contact.SHELL_THICK	Launches the penetration edit panel with the thickest always option for duplicate shells.
Contact.SHELL_THIN	Launches the penetration edit panel with the thinnest always option for duplicate shells.

Contact properties

Name	Type	Description
bt	real	Contact birth time
check_mode	integer	Checking mode on the pen check edit panel. (Can be Contact.MPP_METHOD, Contact.MPP_METHOD or Contact.SMP_METHOD)
cid	integer	Contact number (identical to label.
colour	Colour	The colour of the contact

contact_penchk_dup_shells	integer	Shell treatment on the pen check edit panel. (Can be Contact.SHELL_AUTO, Contact.SHELL_AUTO or Contact.SHELL_THIN or Contact.SHELL_THICK)
dc	real	Exponential decay coeff
dt	real	Contact death time
exists (read only)	logical	true if contact exists, false if referred to but not defined.
fd	real	Dynamic coeff of friction
fs	real	Static coeff of friction
fsf	real	Coulomb friction scale factor
heading	string	Contact heading
id	logical	true if _ID option is set, false if not
include	integer	The Include file number that the contact is in.
label	integer	Contact number.
model (read only)	integer	The Model number that the contact is in.
mortar	logical	_MORTAR keyword option - true if set, false if not.
offset_flag	integer	_OFFSET option. (Can be Contact.NO_OFFSET, Contact.SIMPLE_OFFSET, Contact.CONSTR_OFFSET or Contact.BEAM_OFFSET)
penchk	integer	Penetration search flag
saboxid	integer	Surface A box id
sapr	integer	Surface A side printout flag
sast	real	Optional surface A side shell thickness
sbboxid	integer	Surface B box id
sbpr	integer	Surface B side printout flag
sbst	real	Optional surface B side shell thickness
sfsa	real	Scale factor on surface A penalty stiffness
sfsat	real	Scale factor on true surface A shell thickness
sfsb	real	Scale factor on surface B penalty stiffness
sfsbt	real	Scale factor on true surface B shell thickness
surfa	integer	Surface A set id
surfatyp	integer	Surface A set type
surfb	integer	Surface B set id
surfbtyp	integer	Surface B set type
type	string	Contact type ("AUTOMATIC_GENERAL", "SINGLE_SURFACE" etc).
vc	real	Coeff for viscous friction
vdc	real	Visous damping coefficient
vsf	real	Viscous friction scale factor

Properties for COMPOSITE/LUBRICATION options

Name	Type	Description
cideta	integer	Curve ID for the viscosity
cidmu	integer	Curve ID for the coefficient of friction

d_comp	real	Composite film thickness
srmodel	integer	Model for shear response
tfail	real	Tensile traction for failure

Properties for CONSTRAINT options

Name	Type	Description
kpf	real	Kinematic partition factor

Properties for CONTRACTION_JOINT options

Name	Type	Description
alpha	real	Key amplitude parameter A
beta	real	Key amplitude parameter B
mtcj	integer	The method option for the gap function
tsvx	real	X component of the free sliding direction T
tsvy	real	Y component of the free sliding direction T
tsvz	real	Z component of the free sliding direction T

Properties for DRAWBEAD options

Name	Type	Description
dbdth	real	Draw bead depth
dbpid	integer	optional Part ID
dfscl	real	Scale factor on lcidrf
eloff	integer	optional element id offset
ending	real	Parameter to define the length of the bead
epm	real	Maximum strain the blank will experience when it passes the bead
epscale	real	Scale factor to weaken the stress-strain curve
lceps	integer	Loadcurve ID for plastic strain vs. parametric coord
lceps2	integer	Loadcurve ID for plastic strain vs. parametric coord (elements moved > offset)
lcidnf	integer	Loadcurve ID for Normal force per unit length
lcidrf	integer	Loadcurve ID for Force due to bending per unit length
numint	integer	#int points along drawbead
offset	real	distance offset
tscale	integer	

Properties for ERODING options

Name	Type	Description
erosop	integer	Erosion/interior node option
iadj	integer	Adjacent matl treatment for solids
isym	integer	Symmetry plane option

Properties for INTERFERENCE options

Name	Type	Description
lcid1	integer	Loadcurve ID for Dyn rel stiffness
lcid2	integer	Loadcurve ID for Transient stiffness

Properties for RIGID options

Name	Type	Description
fcm	integer	Force calculation method
lcid	integer	Loadcurve ID for Force vs penetration curve
us	real	Optional unloading stiffness

Properties for THERMAL options

Name	Type	Description
a	integer	Loadcurve ID for a
algo	integer	contact algorithm
b	integer	Loadcurve ID for b
bc_flg	integer	boundary condition flag
c	integer	Loadcurve ID for c
d	integer	Loadcurve ID for d
formula	integer	formula id
frad	real	Radiation conductance across gap
ftosa	real	Fraction of sliding friction energy partitioned to surface A
h0	real	Heat transfer coefficient
k	real	Conductivity of gap fluid
lcfdt	integer	Loadcurve ID for dynamic friction vs. temp
lcfst	integer	Loadcurve ID for static friction vs. temp
lch	integer	Loadcurve ID for lch
lmax	real	Max size for thermal contact
lmin	real	Critical gap size
thermal	logical	If _THERMAL option is set. Can be true or false

Properties for TIEBREAK options

Name	Type	Description
cn	real	Normal stiffness
ct2cn	real	Ratio of tangential stiffness to normal stiffness
eraten	real	Normal energy release rate used in damage calculation
erates	real	Shear energy release rate used in damage calculation
mes	real	Shear force exponent
nen	real	Normal force exponent

nfls	real	Normal failure stress
option	integer	Response option
param	real	Critical distance
sfls	real	Shear failure stress
tblcid	integer	Loadcurve ID for stress vs gap post failure
thkoff	integer	flag for thickness offset

Properties for TIEBREAK_USER options

Name	Type	Description
cn	real	Normal stiffness
ct2cn	real	Ratio of tangential to normal stiff
nhv	integer	Number of history variables
offset	integer	Flag for offset treatment. This is only valid for *CONTACT_AUTOMATIC(_ONE_WAY)_ SURFACE_TO_SURFACE_TIEBREAK_USER and should not be confused with the 'offset' property for other contact types.
option	integer	User tiebreak type
up1	real	User parameter
up10	real	User parameter
up11	real	User parameter
up12	real	User parameter
up13	real	User parameter
up14	real	User parameter
up15	real	User parameter
up16	real	User parameter
up2	real	User parameter
up3	real	User parameter
up4	real	User parameter
up5	real	User parameter
up6	real	User parameter
up7	real	User parameter
up8	real	User parameter
up9	real	User parameter

Properties for TIED_WELD options

Name	Type	Description
close	real	Surface closeness parameter
hclose	real	Thermal contact conductivity
nmhis	integer	Number of material history variables
nmtwh	integer	Number of surface B tied weld history variables
nstwh	integer	Number of surface A tied weld history variables

ntprm	integer	Number of user tied weld parameters
temp	real	Minimum temperature required.
time	real	Minimum time required

Properties for _MPP option

Name	Type	Description
bucket	integer	Bucket sorting frequency
chksegs	integer	Special check for inverted elements
cparm8	integer	Exclude beam to beam contact flag
grpable	integer	Experimental contact algorithm
inititer	integer	Number of iterations for initial penetration checking
lcbucket	integer	Bucket sorting frequency loadcurve ID
mpp	logical	true if _MPP option is set, false if not
ns2track	integer	Number of segments to track per surface A node
parmax	real	The parametric extension distance for contact segments
pensf	real	Ignore penetration scale factor

Properties for optional card A

Name	Type	Description
bsort	integer	Loadcurve for #cycles between bucket sorts
depth	integer	Loadcurve for search depth in automatic contact
frcfrq	integer	#cycles between penalty force updates
lcidab	integer	Loadcurve ID for airbag thickness vs time
maxpar	real	Max parametric coord overlap
sbopt	real	segment based contact option
sofscl	real	Soft constraint scale factor
soft	integer	Soft constraint flag

Properties for optional card B

Name	Type	Description
i2d3d	integer	Segment searching option
isym	integer	Symmetry plane option
penmax	real	Max pen distance for "old" types 3, 5, 10
shlthk	integer	Thickness consideration flag
sldstf	real	Optional solid stiffness
sldthk	real	Optional solid thickness
snlog	integer	Shooting node logic flag
thkopt	integer	Thickness option for "old" types 3, 5, 10

Properties for optional card C

Name	Type	Description
cid_rcf	integer	Coordinate system ID to output rcforc force resultants and ncforc data in a local system
dprfac	real	Depth of penetration reduction factor
dtstif	real	Timestep used in stiffness calc
edgek	real	Scale factor for penalty stiffness of edge-to-edge contact
flangl	real	Angle tolerance in radians for feature lines option in smooth contact
igap	integer	Implicit convergence flag
ignore	integer	Ignore initial pens in automatic types

Properties for optional card D

Name	Type	Description
dnlscl	real	Distance for nonlinear force scaling
dtpchk	real	Time interval between penetration reports
fnlscl	real	Scale factor for nonlinear force scaling
q2tri	integer	Split quads into 2 trias
sfnbr	real	Scale factor for neighbour segment contact
shledg	integer	Edge shape for shells when measuring penetration
tcso	integer	Segment treatment only flag
tiedid	integer	Incremental displacement update for tied contacts

Properties for optional card E

Name	Type	Description
cparm8smp	integer	Spotweld beam flag for SMP
fricsf	real	Scale factor for frictional stiffness
ftorq	integer	Controls the transmittal of moments across the contact interface for some contact types
icor	integer	coefficient of restitution expressed as a percentage
ipback	integer	Create backup penalty tied contact
region	integer	Region to limit contact volume
sharec	integer	Shared constraint flag
srnde	integer	Flag for non-extended exterior shell edges

Properties for optional card F

Name	Type	Description
dbinr	integer	2dbinr - Flag to include 2d belt elements in contact (note properties cannot start with a number, so 2 has been removed).
fstol	real	Tolerance for determining flat segments.
ignroff	integer	Flag to ignore the thickness offset for shells in the calculation of the shell contact penetration depth.
pstiff	integer	Flag to choose the method for calculating the penalty stiffness
ssftyp	integer	Flag to determine how the SSF option on *PART_CONTACT behaves when SOFT = 2 on optional card A

swtpr	integer	Flag to use tapered shell contact segments adjacent to segments that are thinned by the SPOTHIN option on *CONTROL_CONTACT
tetfac	real	Scale factor for the computed volume of tetrahedral solid elements for the mass calculation in SOFT = 2 contact..

Properties for optional card G

Name	Type	Description
cspothin	real	Contact-specific spot weld thinning scale factor to override the scale factor defined in the SPOTHIN option on *CONTROL_CONTACT.
shloff	real	Flag affecting the location of the contact surfaces.

Detailed Description

The Contact class allows you to create, modify, edit and contact cards. See the documentation below for more details.

Constructor

new Contact(Model*[Model]*, type*[string]*, id (optional)*[integer]*, heading (optional)*[string]*)

Description

Create a new Contact object.

Arguments

- **Model** (Model)

Model that Contact will be created in

- **type** (string)

Type of contact

- **id (optional)** (integer)

Contact number

- **heading (optional)** (string)

Title for the Contact

Returns

Contact object

Return type

Contact

Example

To create a new AUTOMATIC_GENERIC contact n model m with label 10 and title "Test contact"

```
var c = new Contact(m, "AUTOMATIC_GENERAL", 10, "Test contact");
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a contact.

Arguments

- **Comment** (Comment)

Comment that will be attached to the contact

Returns

No return value

Example

To associate comment c to the contact c:

```
c.AssociateComment(c);
```

Blank()

Description

Blanks the contact

Arguments

No arguments

Returns

No return value

Example

To blank contact c:

```
c.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the contacts in the model.

Arguments

- **Model** (Model)

Model that all contacts will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the contacts in model m:

```
Contact.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged contacts in the model.

Arguments

- **Model** (Model)

Model that all the flagged contacts will be blanked in

- **flag** (Flag)

Flag set on the contacts that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the contacts in model m flagged with f:

```
Contact.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the contact is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if contact c is blanked:

```
if (c.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse contact c:

```
c.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the contact.

Arguments

- **flag** (Flag)

Flag to clear on the contact

Returns

No return value

Example

To clear flag f for contact c:

```
c.ClearFlag(f);
```

Constrained(connection (optional)*[boolean]*)

Description

see if tied/spotweld contact uses constrained formulation

Arguments

- **connection (optional)** (boolean)

if true will only consider contacts used for PRIMER connections. The default is false.

Returns

logical

Return type

Boolean

Example

To see if contact is of type tied and constrained

```
c.Constrained();
```

Copy(range (optional)*[boolean]*)

Description

Copies the contact. The target include of the copied contact can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

Contact object

Return type

Contact

Example

To copy contact c into contact z:

```
var z = c.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a contact

Arguments

- **Model** (Model)

Model that the contact will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

Contact object (or null if not made).

Return type

Contact

Example

To start creating a contact in model m:

```
var c = Contact.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a contact.

Arguments

- **Comment** (Comment)

Comment that will be detached from the contact

Returns

No return value

Example

To detach comment c from the contact c:

```
c.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit contact c:

```
c.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for contact with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The contact object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for contact c:

```
c.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for contact c:

```
c.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for contact. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for contact c:

```
c.Error("My custom error");
```

ExtractColour()

Description

Extracts the **actual** colour used for contact.
By default in PRIMER many entities such as elements get their colour automatically from the part that they are in. PRIMER cycles through 13 default colours based on the label of the entity. In this case the contact colour property will return the value Colour.PART instead of the actual colour. This method will return the actual colour which is used for drawing the contact.

Arguments

No arguments

Returns

colour value (integer)

Return type

Number

Example

To return the colour used for drawing contact c:

```
var colour = c.ExtractColour();
```

First(Model*[Model]*) [static]

Description

Returns the first contact in the model.

Arguments

- **Model** (Model)

Model to get first contact in

Returns

Contact object (or null if there are no contacts in the model).

Return type

Contact

Example

To get the first contact in model m:

```
var c = Contact.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free contact label in the model. Also see Contact.LastFreeLabel(), Contact.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free contact label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

Contact label.

Return type

Number

Example

To get the first free contact label in model m:

```
var label = Contact.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the contacts in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all contacts will be flagged in

- **flag** (Flag)

Flag to set on the contacts

Returns

No return value

Example

To flag all of the contacts with flag f in model m:

```
Contact.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the contact is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the contact

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if contact c has flag f set on it:

```
if (c.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each contact in the model.
Note that ForEach has been designed to make looping over contacts as fast as possible and so has some limitations.
Firstly, a single temporary Contact object is created and on each function call it is updated with the current contact data. This means that you should not try to store the Contact object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new contacts inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all contacts are in

- **func** (function)

Function to call for each contact

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the contacts in model m:

```
Contact.ForEach(m, test);
function test(c)
{
// c is Contact object
}
```

To call function test for all of the contacts in model m with optional object:

```
var data = { x:0, y:0 };
Contact.ForEach(m, test, data);
function test(c, extra)
{
// c is Contact object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Contact objects or properties for all of the contacts in a model in PRIMER. If the optional property argument is not given then an array of Contact objects is returned. If the property argument is given, that property value for each contact is returned in the array instead of a Contact object

Arguments

- **Model** (Model)

Model to get contacts from

- **property (optional)** (string)

Name for property to get for all contacts in the model

Returns

Array of Contact objects or properties

Return type

Array

Example

To make an array of Contact objects for all of the contacts in model m:

```
var a = Contact.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each contact in model m:

```
var a = Contact.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a contact.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the contact c:

```
var comm_array = c.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of Contact objects for all of the flagged contacts in a model in PRIMER If the optional property argument is not given then an array of Contact objects is returned. If the property argument is given, then that property value for each contact is returned in the array instead of a Contact object

Arguments

- **Model** (Model)

Model to get contacts from

- **flag** (Flag)

Flag set on the contacts that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged contacts in the model

Returns

Array of Contact objects or properties

Return type

Array

Example

To make an array of Contact objects for all of the contacts in model m flagged with f:

```
var c = Contact.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the contacts in model m flagged with f:

```
var a = Contact.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the Contact object for a contact ID.

Arguments

- **Model** (Model)

Model to find the contact in

- **number** (integer)

number of the contact you want the Contact object for

Returns

Contact object (or null if contact does not exist).

Return type

Contact

Example

To get the Contact object for contact 100 in model m

```
var c = Contact.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a Contact property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the Contact.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

contact property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if Contact property c.example is a parameter:

```
Options.property_parameter_names = true;
if (c.GetParameter(c.example) ) do_something...
Options.property_parameter_names = false;
```

To check if Contact property c.example is a parameter by using the GetParameter method:

```
if (c.ViewParameters().GetParameter(c.example) ) do_something...
```

Interactions(type (optional)*[constant]*)

Description

Returns an array of objects describing the interactions which can either be penetrations (tracked nodes that are tied to or penetrate elements in the contact) or crossed edges (contact segments that cross).

Arguments

- **type (optional)** (constant)

What type of interactions to return. Can be bitwise code of Contact.PENETRATIONS to return penetrations and Contact.CROSSED_EDGES to return crossed edges. If omitted penetrations will be returned.

Returns

Array of objects with the following properties:

Name	Type	Description
end	Array of reals	End coordinate of intersection line (for Contact.CROSSED_EDGES)
ex	real	escape vector X component (for Contact.PENETRATIONS)
ey	real	escape vector Y component (for Contact.PENETRATIONS)
ez	real	escape vector Z component (for Contact.PENETRATIONS)
n1	Node object	Node 1 of reference segment (for Contact.PENETRATIONS)
n2	Node object	Node 2 of reference segment (for Contact.PENETRATIONS)
n3	Node object	Node 3 of reference segment (for Contact.PENETRATIONS)
n4	Node object	Node 4 of reference segment (for Contact.PENETRATIONS)
node	Node object	Penetrating node (for Contact.PENETRATIONS)
pen	real	Depth of penetration (for Contact.PENETRATIONS)
qthick	real	Remaining thickness ratio (for Contact.PENETRATIONS)
rthick	real	Remaining unpenetrated thickness (for Contact.PENETRATIONS)
s	real	s parametric coordinate of the tracked node projected onto the shell (for Contact.PENETRATIONS)
shell	Shell object	Penetrated shell (for Contact.PENETRATIONS)
shell1	Shell object	First segment if shell (for Contact.CROSSED_EDGES)
shell2	Shell object	Second segment if shell (for Contact.CROSSED_EDGES)
solid	Solid object	Penetrated solid (for Contact.PENETRATIONS)
solid1	Solid object	First segment if solid (for Contact.CROSSED_EDGES)
solid2	Solid object	Second segment if solid (for Contact.CROSSED_EDGES)
start	Array of reals	Start coordinate of intersection line (for Contact.CROSSED_EDGES)
t	real	t parametric coordinate of the tracked node projected onto the shell (for Contact.PENETRATIONS)
thick	real	Thickness of contact segment, i.e. 0.5*(t1+t2) (for Contact.PENETRATIONS)
thickshell	Tshell object	Penetrated thick shell (for Contact.PENETRATIONS)
thickshell1	Tshell object	First segment if thick shell (for Contact.CROSSED_EDGES)
thickshell2	Tshell object	Second segment if thick shell (for Contact.CROSSED_EDGES)
type	integer	The interaction type. Either Contact.PENETRATIONS or Contact.CROSSED_EDGES.

Return type

object

Example

To get the penetration interactions for contact c:

```
var interactions = c.Interactions();
for(i=0; i<interactions.length; i++)
{
    var type  = interactions[i].type; // Will be Contact.PENETRATIONS
    var node  = interactions[i].node;
    var shell = interactions[i].shell;
    var n1    = interactions[i].n1;
    var n2    = interactions[i].n2;
    var n3    = interactions[i].n3;
    var n4    = interactions[i].n4;
    var s     = interactions[i].s;
    var t     = interactions[i].t;
    var ex    = interactions[i].ex;
    var ey    = interactions[i].ey;
    var ez    = interactions[i].ez;
    var pen   = interactions[i].pen;
    var thick = interactions[i].thick;
    var rthick = interactions[i].rthick;
    var qthick = interactions[i].qthick;
    if(shell != undefined)
            ... process shell ...
}
```

To get the penetration and crossed edge interactions for contact c:

```
var interactions = c.Interactions(Contact.PENETRATIONS|Contact.CROSSED_EDGES);
for(i=0; i<interactions.length; i++)
{
    if (interactions[i].type == Contact.PENETRATIONS)
    {
        var node  = interactions[i].node;
        ...
    }
    else if (interactions[i].type == Contact.CROSSED_EDGES)
    {
        var start = interactions[i].start;
        ...
    }
}
```

Keyword()

Description

Returns the keyword for this Contact (*BOUNDARY_PRESCRIBED_MOTION_xxxx). **Note that a carriage return is not added**. See also Contact.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for Contact c:

```
var key = c.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the Contact. **Note that a carriage return is not added**. See also Contact.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for Contact c:

```
var cards = c.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last contact in the model.

Arguments

- **Model** (Model)

Model to get last contact in

Returns

Contact object (or null if there are no contacts in the model).

Return type

Contact

Example

To get the last contact in model m:

```
var c = Contact.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free contact label in the model. Also see Contact.FirstFreeLabel(), Contact.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free contact label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

Contact label.

Return type

Number

Example

To get the last free contact label in model m:

```
var label = Contact.LastFreeLabel(m);
```

Next()

Description

Returns the next contact in the model.

Arguments

No arguments

Returns

Contact object (or null if there are no more contacts in the model).

Return type

Contact

Example

To get the contact in model m after contact c:

```
var c = c.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) contact label in the model. Also see Contact.FirstFreeLabel(), Contact.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free contact label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

Contact label.

Return type

Number

Example

To get the next free contact label in model m:

```
var label = Contact.NextFreeLabel(m);
```

PenCheck(flag*[Flag]*, eflag*[integer]*)

Description

Flags nodes that penetrate (or tie) in contact

Arguments

- **flag** (Flag)

Flag to be set on penetrating (or tied) node.

- **eflag** (integer)

Optional flag for elements. If supplied, node will be flagged only if it penetrates (or ties to) an element that is flagged. Node and element flag may be the same.

Returns

zero if contact successfully checked

Return type

Number

Example

To set flag f on tracked nodes of Contact c which tie to elements flagged with f:

```
c.PenCheck(f, f);
```

PenCheckEdit(modal (optional)*[boolean]*, check_mode (optional)*[constant]*, mpp_threshold (optional)*[real]*, report_crossed_3d_elems (optional)*[boolean]*, contact_penchk_dup_shells (optional)*[constant]*)

Description

launches the interactive edit panel for penetration check on the con

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

- **check_mode (optional)** (constant)

Check mode. Can be Model.MPP_MODE or Model.SMP_MODE. Default is set to the oa pref contact_check_mode

- **mpp_threshold (optional)** (real)

Can set the MPP threshold, by default this is set to the oa pref contact_mpp_penetration_threshold

- **report_crossed_3d_elems (optional)** (boolean)

Can set the value of reporting crossed elements to TRUE or FALSE, by default this is set to the oa pref report_crossed_3d_elems

- **contact_penchk_dup_shells (optional)** (constant)

Duplicate shell treatment Can be Model.SHELL_AUTO, Model.SHELL_THICK or Model.SHELL_THIN. Default is set to the oa pref contact_penchk_dup_shells

Returns

No return value

Example

To launch an edit panel with modal set to TRUE, check_method set to MPP, mpp_threshold set to 1.123, report_crossed_3d_elems set to true and contact_penchk_dup_shells set to thinnest always:

```
c.PenCheckEdit(true,Contact.MPP_METHOD,1.123,true,Contact.SHELL_THIN);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a contact.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only contacts from that model can be picked. If the argument is a Flag then only contacts that are flagged with *limit* can be selected. If omitted, or null, any contacts from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

Contact object (or null if not picked)

Return type

Contact

Example

To pick a contact from model m giving the prompt 'Pick contact from screen':

```
var c = Contact.Pick('Pick contact from screen', m);
```

Previous()

Description

Returns the previous contact in the model.

Arguments

No arguments

Returns

Contact object (or null if there are no more contacts in the model).

Return type

Contact

Example

To get the contact in model m before contact c:

```
var c = c.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the contacts in the model.

Arguments

- **Model** (Model)

Model that all contacts will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the contacts in model m, from 1000000:

```
Contact.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged contacts in the model.

Arguments

- **Model** (Model)

Model that all the flagged contacts will be renumbered in

- **flag** (Flag)

Flag set on the contacts that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the contacts in model m flagged with f, from 1000000:

```
Contact.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select contacts using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting contacts

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only contacts from that model can be selected. If the argument is a Flag then only contacts that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any contacts can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of contacts selected or null if menu cancelled

Return type

Number

Example

To select contacts from model m, flagging those selected with flag f, giving the prompt 'Select contacts':

```
Contact.Select(f, 'Select contacts', m);
```

To select contacts, flagging those selected with flag f but limiting selection to contacts flagged with flag l, giving the prompt 'Select contacts':

```
Contact.Select(f, 'Select contacts', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the contact.

Arguments

- **flag** (Flag)

Flag to set on the contact

Returns

No return value

Example

To set flag f for contact c:

```
c.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the contact. The contact will be sketched until you either call Contact.Unsketch(), Contact.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the contact is sketched. If omitted redraw is true. If you want to sketch several contacts and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch contact c:

```
c.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged contacts in the model. The contacts will be sketched until you either call Contact.Unsketch(), Contact.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged contacts will be sketched in

- **flag** (Flag)

Flag set on the contacts that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the contacts are sketched. If omitted redraw is true. If you want to sketch flagged contacts several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all contacts flagged with flag in model m:

```
Contact.SketchFlagged(m, flag);
```

StatusCheck()

Description

Checks sliding contact for crossed edges and penetrations

Arguments

No arguments

Returns

An array containing count of crossed edges, count of penetrations (note if a node penetrates more than one segment, it is only reported once here)

Return type

Array

Example

To check Contact c:

```
var status = c.StatusCheck(); ncrossed = status[0]; npens = status[1]
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of contacts in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing contacts should be counted. If false or omitted referenced but undefined contacts will also be included in the total.

Returns

number of contacts

Return type

Number

Example

To get the total number of contacts in model m:

```
var total = Contact.Total(m);
```

Unblank()

Description

Unblanks the contact

Arguments

No arguments

Returns

No return value

Example

To unblank contact c:

```
c.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the contacts in the model.

Arguments

- **Model** (Model)

Model that all contacts will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the contacts in model m:

```
Contact.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged contacts in the model.

Arguments

- **Model** (Model)

Model that the flagged contacts will be unblanked in

- **flag** (Flag)

Flag set on the contacts that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the contacts in model m flagged with f:

```
Contact.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the contacts in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all contacts will be unset in

- **flag** (Flag)

Flag to unset on the contacts

Returns

No return value

Example

To unset the flag f on all the contacts in model m:

```
Contact.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the contact.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the contact is unsketched. If omitted redraw is true. If you want to unsketch several contacts and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch contact c:

```
c.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all contacts.

Arguments

- **Model** (Model)

Model that all contacts will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the contacts are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all contacts in model m:

```
Contact.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged contacts in the model.

Arguments

- **Model** (Model)

Model that all contacts will be unsketched in

- **flag** (Flag)

Flag set on the contacts that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the contacts are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all contacts flagged with flag in model m:

```
Contact.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

Contact object.

Return type

Contact

Example

To check if Contact property c.example is a parameter by using the Contact.GetParameter() method:

```
if (c.ViewParameters().GetParameter(c.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for contact with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description

autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The contact object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for contact c:

```
c.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for contact c:

```
c.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for contact. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for contact c:

```
c.Warning("My custom warning");
```

WriteToExcel(flag*[Flag]*, filename*[string]*) [static]

Description

Writes out all flagged contacts to an Excel file

Arguments

- **flag** (Flag)

Flag set on the contacts that you want to output properties for. Note that this only works for CONT (Sliding Contact)

entities at the moment.

- **filename** (string)

Filename of the Excel file you want to write. If no .xlsx extension is given, an .xlsx extension will be appended to the filename.
The file will be overwritten if it already exists.

Returns

Array of objects

Return type

Array

Example

To write an Excel file ContProps.xlsx for contacts flagged with f:

```
Contact.WriteToExcel(f, "ContProps.xlsx");
```

Xrefs()

Description

Returns the cross references for this contact.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for contact c:

```
var xrefs = c.Xrefs();
```

---

toString()

Description

Creates a string containing the Contact data in keyword format. Note that this contains the keyword header and the keyword cards. See also Contact.Keyword() and Contact.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for Contact c in keyword format

```
var data = c.toString();
```

ContactGuidedCable class

The ContactGuidedCable class gives you access to define *CONTACT_GUIDED_CABLE cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

ContactGuidedCable constants

Name	Description
ContactGuidedCable.PART	CONTACT is *CONTACT_GUIDED_CABLE.
ContactGuidedCable.SET_PART	CONTACT is *CONTACT_GUIDED_CABLE_SET.

ContactGuidedCable properties

Name	Type	Description
cid	integer	ContactGuidedCable number.
endtol	real	Tolerance, in length units.
exists (read only)	logical	true if contact guided_cable exists, false if referred to but not defined.
fric	real	Contact friction.
heading	string	ContactGuidedCable heading
id	logical	TRUE if _ID option is set, FALSE if not
include	integer	The Include file number that the contact guided_cable is in.
model (read only)	integer	The Model number that the contact guided_cable is in.
nsid	integer	Node Set ID that guides the 1D elements.
pid	integer	Part ID or Part Set ID
ptype	constant	The Contact Part type. Can be ContactGuidedCable.PART or ContactGuidedCable.SET_PART.
soft	integer	Flag for soft constraint option. Set to 1 for soft constraint.
ssfac	real	Stiffness scale factor for penalty stiffness value. The default value is unity. This applies to SOFT set to 0 and 1.

Detailed Description

The ContactGuidedCable class allows you to create, modify, edit and manipulate *CONTACT_GUIDED_CABLE cards. See the documentation below for more details.

Constructor

new ContactGuidedCable(Model*[Model]*, ptype*[constant]*, nsid*[integer]*, pid*[integer]*, soft (optional)*[integer]*, ssfac (optional)*[real]*, fric (optional)*[real]*, cid (optional)*[integer]*, heading (optional)*[string]*, endtol (optional)*[real]*)

Description

Create a new ContactGuidedCable object.

Arguments

- **Model** (Model)

Model that ContactGuidedCable will be created in

- **ptype** (constant)

Specify the type of ContactGuidedCable (Can be ContactGuidedCable.PART or ContactGuidedCable.SET_PART

- **nsid** (integer)

Node Set ID that guides the 1D elements.

- **pid** (integer)

Part ID or Part Set ID

- **soft (optional)** (integer)

Flag for soft constraint option. Set to 1 for soft constraint.

- **ssfac (optional)** (real)

Stiffness scale factor for penalty stiffness value. The default value is unity. This applies to SOFT set to 0 and 1.

- **fric (optional)** (real)

Contact friction.

- **cid (optional)** (integer)

ContactGuidedCable number (Same as label).

- **heading (optional)** (string)

ContactGuidedCable heading (Same as title).

- **endtol (optional)** (real)

Tolerance, in length units.

Returns

ContactGuidedCable object

Return type

ContactGuidedCable

Example

To create a new contact guided_cable in model m, of ptype PART, with nsid 100, pid 10, soft 1 and ssfac 4.5.

```
var c_g_c = new ContactGuidedCable(m, ContactGuidedCable.PART, 100, 10, 1, 4.5);
```

Details of functions

AssociateComment(Comment[*Comment*])

Description

Associates a comment with a contact guided_cable.

Arguments

- **Comment** (Comment)

Comment that will be attached to the contact guided_cable

Returns

No return value

Example

To associate comment c to the contact guided_cable c_g_c:

```
c_g_c.AssociateComment(c);
```

Blank()

Description

Blanks the contact guided_cable

Arguments

No arguments

Returns

No return value

Example

To blank contact guided_cable c_g_c:

```
c_g_c.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the contact guided_cables in the model.

Arguments

- **Model** (Model)

Model that all contact guided_cables will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the contact guided_cables in model m:

```
ContactGuidedCable.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged contact guided_cables in the model.

Arguments

- **Model** (Model)

Model that all the flagged contact guided_cables will be blanked in

- **flag** (Flag)

Flag set on the contact guided_cables that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the contact guided_cables in model m flagged with f:

```
ContactGuidedCable.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the contact guided_cable is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if contact guided_cable c_g_c is blanked:

```
if (c_g_c.Blanked() ) do_something...
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the contact guided_cable.

Arguments

- **flag** (Flag)

Flag to clear on the contact guided_cable

Returns

No return value

Example

To clear flag f for contact guided_cable c_g_c:

```
c_g_c.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the contact guided_cable. The target include of the copied contact guided_cable can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

ContactGuidedCable object

Return type

ContactGuidedCable

Example

To copy contact guided_cable c_g_c into contact guided_cable z:

```
var z = c_g_c.Copy();
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a contact guided_cable.

Arguments

- **Comment** (Comment)

Comment that will be detached from the contact guided_cable

Returns

No return value

Example

To detach comment c from the contact guided_cable c_g_c:

```
c_g_c.DetachComment(c);
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for contact guided_cable with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description

autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The contact guided_cable object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for contact guided_cable c_g_c:

```
c_g_c.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for contact guided_cable c_g_c:

```
c_g_c.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for contact guided_cable. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for contact guided_cable c_g_c:

```
c_g_c.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first contact guided_cable in the model.

Arguments

- **Model** (Model)

Model to get first contact guided_cable in

Returns

ContactGuidedCable object (or null if there are no contact guided_cables in the model).

Return type

ContactGuidedCable

Example

To get the first contact guided_cable in model m:

```
var c_g_c = ContactGuidedCable.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free contact guided_cable label in the model. Also see ContactGuidedCable.LastFreeLabel(), ContactGuidedCable.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free contact guided_cable label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

ContactGuidedCable label.

Return type

Number

Example

To get the first free contact guided_cable label in model m:

```
var label = ContactGuidedCable.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the contact guided_cables in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all contact guided_cables will be flagged in

- **flag** (Flag)

Flag to set on the contact guided_cables

Returns

No return value

Example

To flag all of the contact guided_cables with flag f in model m:

```
ContactGuidedCable.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the contact guided_cable is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the contact guided_cable

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if contact guided_cable c_g_c has flag f set on it:

```
if (c_g_c.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each contact guided_cable in the model.
Note that ForEach has been designed to make looping over contact guided_cables as fast as possible and so has some limitations.
Firstly, a single temporary ContactGuidedCable object is created and on each function call it is updated with the current contact guided_cable data. This means that you should not try to store the ContactGuidedCable object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new contact guided_cables inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all contact guided_cables are in

- **func** (function)

Function to call for each contact guided_cable

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the contact guided_cables in model m:

```
ContactGuidedCable.ForEach(m, test);
function test(c_g_c)
{
// c_g_c is ContactGuidedCable object
}
```

To call function test for all of the contact guided_cables in model m with optional object:

```
var data = { x:0, y:0 };
ContactGuidedCable.ForEach(m, test, data);
function test(c_g_c, extra)
{
// c_g_c is ContactGuidedCable object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of ContactGuidedCable objects or properties for all of the contact guided_cables in a model in PRIMER. If the optional property argument is not given then an array of ContactGuidedCable objects is returned. If the property argument is given, that property value for each contact guided_cable is returned in the array instead of a ContactGuidedCable object

Arguments

- **Model** (Model)

Model to get contact guided_cables from

- **property (optional)** (string)

Name for property to get for all contact guided_cables in the model

Returns

Array of ContactGuidedCable objects or properties

Return type

Array

Example

To make an array of ContactGuidedCable objects for all of the contact guided_cables in model m:

```
var a = ContactGuidedCable.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each contact guided_cable in model m:

```
var a = ContactGuidedCable.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a contact guided_cable.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the contact guided_cable c_g_c:

```
var comm_array = c_g_c.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of ContactGuidedCable objects for all of the flagged contact guided_cables in a model in PRIMER If the optional property argument is not given then an array of ContactGuidedCable objects is returned. If the property argument is given, then that property value for each contact guided_cable is returned in the array instead of a ContactGuidedCable object

Arguments

- **Model** (Model)

Model to get contact guided_cables from

- **flag** (Flag)

Flag set on the contact guided_cables that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged contact guided_cables in the model

Returns

Array of ContactGuidedCable objects or properties

Return type

Array

Example

To make an array of ContactGuidedCable objects for all of the contact guided_cables in model m flagged with f:

```
var c_g_c = ContactGuidedCable.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the contact guided_cables in model m flagged with f:

```
var a = ContactGuidedCable.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the ContactGuidedCable object for a contact guided_cable ID.

Arguments

- **Model** (Model)

Model to find the contact guided_cable in

- **number** (integer)

number of the contact guided_cable you want the ContactGuidedCable object for

Returns

ContactGuidedCable object (or null if contact guided_cable does not exist).

Return type

ContactGuidedCable

Example

To get the ContactGuidedCable object for contact guided_cable 100 in model m

```
var c_g_c = ContactGuidedCable.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a ContactGuidedCable property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the ContactGuidedCable.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

contact guided_cable property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if ContactGuidedCable property c_g_c.example is a parameter:

```
Options.property_parameter_names = true;
if (c_g_c.GetParameter(c_g_c.example) ) do_something...
Options.property_parameter_names = false;
```

To check if ContactGuidedCable property c_g_c.example is a parameter by using the GetParameter method:

```
if (c_g_c.ViewParameters().GetParameter(c_g_c.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this ContactGuidedCable (*contact_guided_cable). **Note that a carriage return is not added**. See also ContactGuidedCable.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for ContactGuidedCable c_g_c:

```
var key = c_g_c.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the ContactGuidedCable. **Note that a carriage return is not added**. See also ContactGuidedCable.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for ContactGuidedCable c_g_c:

```
var cards = c_g_c.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last contact guided_cable in the model.

Arguments

- **Model** (Model)

Model to get last contact guided_cable in

Returns

ContactGuidedCable object (or null if there are no contact guided_cables in the model).

Return type

ContactGuidedCable

Example

To get the last contact guided_cable in model m:

```
var c_g_c = ContactGuidedCable.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free contact guided_cable label in the model. Also see ContactGuidedCable.FirstFreeLabel(), ContactGuidedCable.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free contact guided_cable label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

ContactGuidedCable label.

Return type

Number

Example

To get the last free contact guided_cable label in model m:

```
var label = ContactGuidedCable.LastFreeLabel(m);
```

Next()

Description

Returns the next contact guided_cable in the model.

Arguments

No arguments

Returns

ContactGuidedCable object (or null if there are no more contact guided_cables in the model).

Return type

ContactGuidedCable

Example

To get the contact guided_cable in model m after contact guided_cable c_g_c:

```
var c_g_c = c_g_c.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) contact guided_cable label in the model. Also see ContactGuidedCable.FirstFreeLabel(), ContactGuidedCable.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free contact guided_cable label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

ContactGuidedCable label.

Return type

Number

Example

To get the next free contact guided_cable label in model m:

```
var label = ContactGuidedCable.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a contact guided_cable.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only contact guided_cables from that model can be picked. If the argument is a Flag then only contact guided_cables that are flagged with *limit* can be selected. If omitted, or null, any contact guided_cables from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

ContactGuidedCable object (or null if not picked)

Return type

ContactGuidedCable

Example

To pick a contact guided_cable from model m giving the prompt 'Pick contact guided_cable from screen':

```
var c_g_c = ContactGuidedCable.Pick('Pick contact guided_cable from screen', m);
```

Previous()

Description

Returns the previous contact guided_cable in the model.

Arguments

No arguments

Returns

ContactGuidedCable object (or null if there are no more contact guided_cables in the model).

Return type

ContactGuidedCable

Example

To get the contact guided_cable in model m before contact guided_cable c_g_c:

```
var c_g_c = c_g_c.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the contact guided_cables in the model.

Arguments

- **Model** (Model)

Model that all contact guided_cables will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the contact guided_cables in model m, from 1000000:

```
ContactGuidedCable.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged contact guided_cables in the model.

Arguments

- **Model** (Model)

Model that all the flagged contact guided_cables will be renumbered in

- **flag** (Flag)

Flag set on the contact guided_cables that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the contact guided_cables in model m flagged with f, from 1000000:

```
ContactGuidedCable.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select contact guided_cables using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting contact guided_cables

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only contact guided_cables from that model can be selected. If the argument is a Flag then only contact guided_cables that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any contact guided_cables can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of contact guided_cables selected or null if menu cancelled

Return type

Number

Example

To select contact guided_cables from model m, flagging those selected with flag f, giving the prompt 'Select contact guided_cables':

```
ContactGuidedCable.Select(f, 'Select contact guided_cables', m);
```

To select contact guided_cables, flagging those selected with flag f but limiting selection to contact guided_cables flagged with flag l, giving the prompt 'Select contact guided_cables':

```
ContactGuidedCable.Select(f, 'Select contact guided_cables', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the contact guided_cable.

Arguments

- **flag** (Flag)

Flag to set on the contact guided_cable

Returns

No return value

Example

To set flag f for contact guided_cable c_g_c:

```
c_g_c.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the contact guided_cable. The contact guided_cable will be sketched until you either call ContactGuidedCable.Unsketch(), ContactGuidedCable.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the contact guided_cable is sketched. If omitted redraw is true. If you want to sketch several contact guided_cables and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch contact guided_cable c_g_c:

```
c_g_c.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged contact guided_cables in the model. The contact guided_cables will be sketched until you either call ContactGuidedCable.Unsketch(), ContactGuidedCable.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged contact guided_cables will be sketched in

- **flag** (Flag)

Flag set on the contact guided_cables that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the contact guided_cables are sketched. If omitted redraw is true. If you want to sketch flagged contact guided_cables several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all contact guided_cables flagged with flag in model m:

```
ContactGuidedCable.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of contact guided_cables in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing contact guided_cables should be counted. If false or omitted referenced but undefined contact guided_cables will also be included in the total.

Returns

number of contact guided_cables

Return type

Number

Example

To get the total number of contact guided_cables in model m:

```
var total = ContactGuidedCable.Total(m);
```

Unblank()

Description

Unblanks the contact guided_cable

Arguments

No arguments

Returns

No return value

Example

To unblank contact guided_cable c_g_c:

```
c_g_c.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the contact guided_cables in the model.

Arguments

- **Model** (Model)

Model that all contact guided_cables will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the contact guided_cables in model m:

```
ContactGuidedCable.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged contact guided_cables in the model.

Arguments

- **Model** (Model)

Model that the flagged contact guided_cables will be unblanked in

- **flag** (Flag)

Flag set on the contact guided_cables that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the contact guided_cables in model m flagged with f:

```
ContactGuidedCable.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the contact guided_cables in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all contact guided_cables will be unset in

- **flag** (Flag)

Flag to unset on the contact guided_cables

Returns

No return value

Example

To unset the flag f on all the contact guided_cables in model m:

```
ContactGuidedCable.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the contact guided_cable.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the contact guided_cable is unsketched. If omitted redraw is true. If you want to unsketch several contact guided_cables and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch contact guided_cable c_g_c:

```
c_g_c.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all contact guided_cables.

Arguments

- **Model** (Model)

Model that all contact guided_cables will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the contact guided_cables are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all contact guided_cables in model m:

```
ContactGuidedCable.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged contact guided_cables in the model.

Arguments

- **Model** (Model)

Model that all contact guided_cables will be unsketched in

- **flag** (Flag)

Flag set on the contact guided_cables that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the contact guided_cables are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all contact guided_cables flagged with flag in model m:

```
ContactGuidedCable.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

ContactGuidedCable object.

Return type

ContactGuidedCable

Example

To check if ContactGuidedCable property c_g_c.example is a parameter by using the ContactGuidedCable.GetParameter() method:

```
if (c_g_c.ViewParameters().GetParameter(c_g_c.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for contact guided_cable with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description

autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The contact guided_cable object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for contact guided_cable c_g_c:

```
c_g_c.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for contact guided_cable c_g_c:

```
c_g_c.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for contact guided_cable. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for contact guided_cable c_g_c:

```
c_g_c.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this contact guided_cable.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for contact guided_cable c_g_c:

```
var xrefs = c_g_c.Xrefs();
```

---

toString()

Description

Creates a string containing the ContactGuidedCable data in keyword format. Note that this contains the keyword header and the keyword cards. See also ContactGuidedCable.Keyword() and ContactGuidedCable.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for ContactGuidedCable c_g_c in keyword format

```
var s = c_g_c.toString();
```

Control class

The Control class gives you access to control cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Control properties

Name	Type	Description
accuracy	Object	*CONTROL_ACCURACY card
acoustic	Object	*CONTROL_ACOUSTIC card
acoustic_coupling	Object	*CONTROL_ACOUSTIC_COUPLING card
acoustic_spectral	Object	*CONTROL_ACOUSTIC_SPECTRAL card
adapstep	Object	*CONTROL_ADAPSTEP card
adaptive	Object	*CONTROL_ADAPTIVE card
adaptive_curve	Object	*CONTROL_ADAPTIVE_CURVE card
airbag	Object	*CONTROL_AIRBAG card
ale	Object	*CONTROL_ALE card
bulk_viscosity	Object	*CONTROL_BULK_VISCOSITY card
check	Object	*CONTROL_CHECK card
coarsen	Object	*CONTROL_COARSEN card
constrained	Object	*CONTROL_CONSTRAINED card
contact	Object	*CONTROL_CONTACT card
coupling	Object	*CONTROL_COUPLING card
cpg	Object	*CONTROL_CPG card
cpm	Object	*CONTROL_CPM card
cpu	Object	*CONTROL_CPU card
debug	Object	*CONTROL_DEBUG card
discrete_element	Object	*CONTROL_DISCRETE_ELEMENT card
dynamic_relaxation	Object	*CONTROL_DYNAMIC_RELAXATION card
efg	Object	*CONTROL_EFG card
energy	Object	*CONTROL_ENERGY card
eos_user_library	Object	*CONTROL_EOS_USER_LIBRARY card
explicit_thermal	Object	*CONTROL_EXPLICIT_THERMAL_PROPERTIES card

explicit_thermal_ale_coupling	Object	*CONTROL_EXPLICIT_THERMAL_ALE_COUPLING card
explicit_thermal_boundary	Object	*CONTROL_EXPLICIT_THERMAL_BOUNDARY card
explicit_thermal_contact	Object	*CONTROL_EXPLICIT_THERMAL_CONTACT card
explicit_thermal_initial	Object	*CONTROL_EXPLICIT_THERMAL_INITIAL card
explicit_thermal_output	Object	*CONTROL_EXPLICIT_THERMAL_OUTPUT card
explicit_thermal_solver	Object	*CONTROL_EXPLICIT_THERMAL_SOLVER card
explosive_shadow	Object	*CONTROL_EXPLOSIVE_SHADOW card
forming_bestfit	Object	*CONTROL_FORMING_BESTFIT card
forming_initial_thickness	Object	*CONTROL_FORMING_INITIAL_THICKNESS card
forming_maxid	Object	*CONTROL_FORMING_MAXID card
forming_position	Object	*CONTROL_FORMING_POSITION card
forming_pre_bending	Object	*CONTROL_FORMING_PRE_BENDING card
forming_projection	Object	*CONTROL_FORMING_PROJECTION card
forming_remove_adaptive_constraints	Object	*CONTROL_FORMING_REMOVE_ADAPTIVE_CONSTRAINTS card
forming_shell_to_tshell	Object	*CONTROL_FORMING_SHELL_TO_TSHELL card
forming_stoning	Object	*CONTROL_FORMING_STONING card
forming_strain_ratio_smooth	Object	*CONTROL_FORMING_STRAIN_RATIO_SMOOTH card
forming_template	Object	*CONTROL_FORMING_TEMPLATE card
forming_toleranc	Object	*CONTROL_FORMING_TOLERANC card
forming_travel	Object	*CONTROL_FORMING_TRAVEL card
forming_trim_merge	Object	*CONTROL_FORMING_TRIM_MERGE card
forming_trim_solid_refinement	Object	*CONTROL_FORMING_TRIM_SOLID_REFINEMENT card
forming_unflanging	Object	*CONTROL_FORMING_UNFLANGING card
forming_user	Object	*CONTROL_FORMING_USER card
frequency_domain	Object	*CONTROL_FREQUENCY_DOMAIN card
frequency_response_function	Object	*CONTROL_FREQUENCY_RESPONSE_FUNCTION card
hourglass	Object	*CONTROL_HOURGLASS card
implicit_auto	Object	*CONTROL_IMPLICIT_AUTO card
implicit_buckle	Object	*CONTROL_IMPLICIT_BUCKLE card
implicit_consistent_mass	Object	*CONTROL_IMPLICIT_CONSISTENT_MASS card
implicit_dynamics	Object	*CONTROL_IMPLICIT_DYNAMICS card
implicit_eigenvalue	Object	*CONTROL_IMPLICIT_EIGENVALUE card
implicit_explicit_hybrid	Object	*CONTROL_IMPLICIT_EXPLICIT_HYBRID card
implicit_forming	Object	*CONTROL_IMPLICIT_FORMING card

implicit_general	Object	*CONTROL_IMPLICIT_GENERAL card
implicit_inertia_relief	Object	*CONTROL_IMPLICIT_INERTIA_RELIEF card
implicit_joints	Object	*CONTROL_IMPLICIT_JOINTS card
implicit_modal_dynamic	Object	*CONTROL_IMPLICIT_MODAL_DYNAMIC card
implicit_modal_dynamic_damping	Object	*CONTROL_IMPLICIT_MODAL_DYNAMIC_DAMPING card
implicit_modes	Object	*CONTROL_IMPLICIT_MODES card
implicit_ordering	Object	*CONTROL_IMPLICIT_ORDERING card
implicit_residual_vector	Object	*CONTROL_IMPLICIT_RESIDUAL_VECTOR card
implicit_solution	Object	*CONTROL_IMPLICIT_SOLUTION card
implicit_solver	Object	*CONTROL_IMPLICIT_SOLVER card
implicit_ssd_direct	Object	*CONTROL_IMPLICIT_SSD_DIRECT card
implicit_stabilization	Object	*CONTROL_IMPLICIT_STABILIZATION card
implicit_static_condensation	Object	*CONTROL_IMPLICIT_STATIC_CONDENSATION card
implicit_termination	Object	*CONTROL_IMPLICIT_TERMINATION card
mapping_shift_by_dt	Object	*CONTROL_MAPPING_SHIFT_BY_DT card
mpp_contact_groupable	Object	*CONTROL_MPP_CONTACT_GROUPABLE card
mpp_decomposition_adjust_mtype_cost	Object	*CONTROL_MPP_DECOMPOSITION_ADJUST_MTYPE_COST card
mpp_decomposition_adjust_part_cost	Object	*CONTROL_MPP_DECOMPOSITION_ADJUST_PART_COST card
mpp_decomposition_automatic	Object	*CONTROL_MPP_DECOMPOSITION_AUTOMATIC card
mpp_decomposition_bagref	Object	*CONTROL_MPP_DECOMPOSITION_BAGREF card
mpp_decomposition_check_speed	Object	*CONTROL_MPP_DECOMPOSITION_CHECK_SPEED card
mpp_decomposition_contact_isolate	Object	*CONTROL_MPP_DECOMPOSITION_CONTACT_ISOLATE card
mpp_decomposition_disable_unref_curves	Object	*CONTROL_MPP_DECOMPOSITION_DISABLE_UNREF_CURVES card
mpp_decomposition_distribute_ale_elements	Object	*CONTROL_MPP_DECOMPOSITION_DISTRIBUTE_ALE_ELEMENTS card
mpp_decomposition_distribute_sph_elements	Object	*CONTROL_MPP_DECOMPOSITION_DISTRIBUTE_SPH_ELEMENTS card
mpp_decomposition_elcost	Object	*CONTROL_MPP_DECOMPOSITION_ELCOST card
mpp_decomposition_file	Object	*CONTROL_MPP_DECOMPOSITION_FILE card
mpp_decomposition_flag_stress_strain_curve	Object	*CONTROL_MPP_DECOMPOSITION_FLAG_STRESS_STRAIN_CURVE card
mpp_decomposition_method	Object	*CONTROL_MPP_DECOMPOSITION_METHOD card
mpp_decomposition_numproc	Object	*CONTROL_MPP_DECOMPOSITION_NUMPROC card
mpp_decomposition_outdecomp	Object	*CONTROL_MPP_DECOMPOSITION_OUTDECOMP card

mpp_decomposition_rcblog	Object	*CONTROL_MPP_DECOMPOSITION_RCBLOG card
mpp_decomposition_scale_contact_cost	Object	*CONTROL_MPP_DECOMPOSITION_SCALE_CONTACT_COST card
mpp_decomposition_scale_factor_sph	Object	*CONTROL_MPP_DECOMPOSITION_SCALE_FACTOR_SPH card
mpp_decomposition_show	Object	*CONTROL_MPP_DECOMPOSITION_SHOW card
mpp_decomposition_transformation	Object	*CONTROL_MPP_DECOMPOSITION_TRANSFORMATION card
mpp_io_binoutonly	Object	*CONTROL_MPP_IO_BINOUTONLY card
mpp_io_lstc_reduce	Object	*CONTROL_MPP_IO_LSTC_REDUCE card
mpp_io_nod3dump	Object	*CONTROL_MPP_IO_NOD3DUMP card
mpp_io_nodump	Object	*CONTROL_MPP_IO_NODUMP card
mpp_io_nofail	Object	*CONTROL_MPP_IO_NOFAIL card
mpp_io_nofull	Object	*CONTROL_MPP_IO_NOFULL card
mpp_io_swapbytes	Object	*CONTROL_MPP_IO_SWAPBYTES card
mpp_mat_model_driver	Object	*CONTROL_MPP_MATERIAL_MODEL_DRIVER card
mpp_rebalance	Object	*CONTROL_MPP_REBALANCE card
nonlocal	Object	*CONTROL_NONLOCAL card
output	Object	*CONTROL_OUTPUT card
parallel	Object	*CONTROL_PARALLEL card
pore_air	Object	*CONTROL_PORE_AIR card
pore_fluid	Object	*CONTROL_PORE_FLUID card
pwp_auto_tmf	Object	*CONTROL_PWP_AUTO_TMF card
pzelectric	Object	*CONTROL_PZELECTRIC card
ref_config	Object	*CONTROL_REFERENCE_CONFIGURATION card
remesh	Object	*CONTROL_REMESHING card
rigid	Object	*CONTROL_RIGID card
shell	Object	*CONTROL_SHELL card
solid	Object	*CONTROL_SOLID card
solution	Object	*CONTROL_SOLUTION card
sph	Object	*CONTROL_SPH card
sph_incompressible	Object	*CONTROL_SPH_INCOMPRESSIBLE card
spotweld_beam	Object	*CONTROL_SPOTWELD_BEAM card
staged_construction	Object	*CONTROL_STAGED_CONSTRUCTION card
start	Object	*CONTROL_START card
steady_state_rolling	Object	*CONTROL_STEADY_STATE_ROLLING card
structured	Object	*CONTROL_STRUCTURED card
termination	Object	*CONTROL_TERMINATION card

thermal_eigenvalue	Object	*CONTROL_THERMAL_EIGENVALUE card
thermal_forming	Object	*CONTROL_THERMAL_FORMING card
thermal_nonlinear	Object	*CONTROL_THERMAL_NONLINEAR card
thermal_solver	Object	*CONTROL_THERMAL_SOLVER card
thermal_timestep	Object	*CONTROL_THERMAL_TIMESTEP card
timestep	Object	*CONTROL_TIMESTEP card
units	Object	*CONTROL_UNITS card
vibro_acoustic	Object	*CONTROL_VIBRO_ACOUSTIC card

Detailed Description

The Control class allows you to create, modify, edit and manipulate control cards. Unlike other classes there is no constructor and there are no functions. Instead a Control object is available as the control property of a Model object. This object allows you to access all of the control cards.

For example, to activate control card *CONTROL_TERMINATION in model m and set endtim to 0.1.

```
m.control.termination.exists = true;
m.control.termination.endtim = 0.1;
```

See the properties for more details.

*CONTROL_ACCURACY

Properties for *CONTROL_ACCURACY

Name	Type	Description
exacc	real	Explicit accuracy parameter
exists	logical	true if control card exists
iacc	integer	Implicit accuracy flag
include	integer	The Include file number that the control card is in.
inn	integer	Invariant node numbering for shell element
osu	integer	Objective stress update for large timestep
pidosu	integer	Part set id for objective stress updates

*CONTROL_ACOUSTIC

Properties for *CONTROL_ACOUSTIC

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
macdvp	logical	Acoustic nodal motions will be calculated or not.

*CONTROL_ACOUSTIC_COUPLING

Properties for *CONTROL_ACOUSTIC_COUPLING

Name	Type	Description
acecf1	real	Multiplier on proximity test.
acecf2	real	Angle between normal vectors in an orientation test.
acecf3	real	Multiplier on ceiling test.
acecf4	real	Area equilibration threshold.
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
maccpl	integer	Coupling method.

*CONTROL_ACOUSTIC_SPECTRAL

Properties for *CONTROL_ACOUSTIC_SPECTRAL

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
masehrf	integer	Optional h-refinement.
maseigx	integer	Approach to element time step calculation.
masekfl	integer	Dump flag for h-refined and spectral element meshes.
maseord	integer	Spectral element integration order.
maseplt	integer	Flag to output a high-resolution plot state form.

*CONTROL_ADAPSTEP

Properties for *CONTROL_ADAPSTEP

Name	Type	Description
dfactr	real	Incremental increase in factin
exists	logical	true if control card exists
factin	real	Initial relaxation factor for contact force
include	integer	The Include file number that the control card is in.

*CONTROL_ADAPTIVE

Properties for *CONTROL_ADAPTIVE

Name	Type	Description
adpass	integer	1 or 2 pass adaptivity flag
adpctl	real	Adaptivity error tolerance in degrees for activating fusion
adpd3p	integer	Flag to control the output of adaptive mesh & d3plot files
adpdam	integer	Type of damage accumulation in the workpiece

adpene	real	Nodal penetration at which to refine elem
adperr	integer	Options for recovery techniques and error estimators
adpfreq	real	Time interval between refinements
adpopt	integer	Adaptive options
adpscl	real	Strain rate scale factor
adpsize	real	Min element edge size for adaptivity
adpth	real	Absolute shell thickness below which remeshing should begin
adptol	real	Adaptive error tolerance (degrees)
adptyp	integer	Adaptive options
cbirth	real	Birth time for adaptive fusion
cdeath	real	Death time for adaptive fusion
cnla	real	Limit angle for corner nodes
d3trace	integer	Flag for writing out D3PLOT state
exists	logical	true if control card exists
iadpcl	integer	Fission level that fusion will start at
iadpe90	integer	Maximum no. of elements covering 90degree of radii
iadpgh	integer	Fiffion flag for neighbour splitting
ifsand	integer	Flag for forming of sandwiched parts with adaptive blank mesh
include	integer	The Include file number that the control card is in.
ioflag	integer	Flag to generate adaptive mesh
ireflg	integer	Uniform refinement level. Loadcurve if negative
lcadp	integer	Loadcurve: Adaptive interval vs time
lclvl	integer	Loadcurve of maximum refinement level vs. time
maxel	integer	Max number of elements for adaptivity
maxlvl	integer	Max #refinement levels
memory	integer	Memory limit beyond which adaptivity will cease
mmm2d	integer	Option for merging common boundaries of all adapted materials
ncfreq	integer	Frequency of fission to fusion steps
orient	integer	Flag to set the global orientation of a forming contact
tbirth	real	Birth time for adaptivity
tdeath	real	Death time for adaptivity

*CONTROL_ADAPTIVE_CURVE

Properties for *CONTROL_ADAPTIVE_CURVE

Name	Type	Description
exists	logical	true if control card exists
idset	integer	Shell/Part set id
include	integer	The Include file number that the control card is in.

itriopt	integer	Refinement option for enclosed area of trim curve
itype	integer	Set type
n	integer	Refinement option
smin	real	Element dimension limit for refining

*CONTROL_AIRBAG

Properties for *CONTROL_AIRBAG

Name	Type	Description
ckerr	integer	Flag to check and report open edge of CV airbag
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.

*CONTROL_ALE

Properties for *CONTROL_ALE

Name	Type	Description
aafac	real	ALE advection factor
afac	real	Smoothing weight factor: simple average
beamin	real	Flag for aligning beam dynamics
bfac	real	Smoothing weight factor: volume weighting
bndflx	integer	Multi-Material ALE group set if positive or -1
cfac	real	Smoothing weight factor: isoparametric
checkr	real	Parameter for ALE pressure locking
dct	integer	Default continuum treatment
dfac	real	Smoothing weight factor: equipotential
dtmufac	real	Scale time step called DTMU
ebc	integer	Automatic Euler boundary condition
efac	real	Smoothing weight factor: equipotential
end	real	End time for smoothing
exists	logical	true if control card exists
ialedr	integer	Include ALE computations in the dynamic relaxation analysis
imascl	integer	Flag for mass scaling for ALE parts
include	integer	The Include file number that the control card is in.
meth	integer	Advection method
minmas	real	Factor of the minimum mass allowed in an element
mmgpref	integer	Selects the method that is used to include a reference pressure in a calculation involving ALE multi-material groups
nadv	integer	Number of cycles between advections

nbkt	integer	Number of Lagrangian cycles between bucket sort searches
ncpl	integer	Number of Lagrangian cycles between coupling calculations
nsidebc	integer	Optional excluded node set
optimpp	integer	Optimize the MPP communications (Range 0/1)
pdifmx	real	Max pressure difference for stress zeroing
pref	real	ref pressure on boundary
prit	integer	Pressure equalibrium flag
start	real	Start time for smoothing
vfact	real	Void factor

*CONTROL_BULK_VISCOSITY

Properties for *CONTROL_BULK_VISCOSITY

Name	Type	Description
btype	integer	beam bulk viscosity type
exists	logical	true if control card exists
ibq	integer	Default bulk viscosity type (m#PR035)
include	integer	The Include file number that the control card is in.
q1	real	Default linear viscosity coefficient
q2	real	Default quadratic viscosity coefficient
tstype	integer	Thick shell bulk viscosity type

*CONTROL_CHECK

Properties for *CONTROL_CHECK

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
solitary	integer	TRUE if a plain (no _SHELL suffix) card exists

*CONTROL_COARSEN

Properties for *CONTROL_COARSEN

Name	Type	Description
angle	real	Permitted angle between neighbours
exists	logical	true if control card exists
icoarse	integer	On/Off flag
include	integer	The Include file number that the control card is in.

n1	integer	Optional seed node ID 1
n2	integer	Optional seed node ID 2
n3	integer	Optional seed node ID 3
n4	integer	Optional seed node ID 4
n5	integer	Optional seed node ID 5
n6	integer	Optional seed node ID 6
n7	integer	Optional seed node ID 7
n8	integer	Optional seed node ID 8
nseed	integer	#extra "seed" nodes below
psid	integer	excluded part set
smax	real	Maximum element size

*CONTROL_CONSTRAINED

Properties for *CONTROL_CONSTRAINED

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
sprchk	integer	Flag to check and report open edge of CV constrained
sprsmd	integer	Shear moment distribution behavior for SPR3
sprsrch	integer	Search method for SPR2 and SPR3

*CONTROL_CONTACT

Properties for *CONTROL_CONTACT

Name	Type	Description
cohtiem	integer	Flag to treat how the mass from SURFB of a tied contact affects the time step estimation of cohesive elements
dfric	real	Default dynamic coefficient of friction
dir_tie	integer	Directional tie for MPP non-groupable tied contacts
ecdt	integer	Timestep override for eroding contacts
edc	real	Default exponential decay coefficient
enmass	integer	Treatment of mass of eroded nodes
exists	logical	true if control card exists
frceng	integer	Flag to calculate internal friction energy
ftall	integer	output contact forces to rcforc
icov	integer	Invokes the covariant formulation of Konyukhov and Schweizerhof
igactc	integer	option to use isogeometric shells for contact detection
ignore	integer	Ignore initial penetrations flag

include	integer	The Include file number that the control card is in.
interm	integer	Intermittent searching flag for old contacts
irevspt	integer	Flag to revert the spot weld thinning behavior
islchk	integer	Initial penetration check flag
isym	integer	symmetry option. Node set if negative
ithcnt	integer	thermal contact heat transfer mode
ithoff	integer	Flag for offsetting thermal contact surfaces for thick thermal shells
nsbcs	integer	#cycles between 3D bucket sorts
nserod	integer	erosion option
orien	integer	Automatic contact segment orientation flag
outseg	integer	Spotweld output flag
pen_sf	real	Default local penalty scale factor
penopt	integer	Penalty stiffness option flag
pstiff	integer	method for penalty stiff calc
ptscl	real	scale factor on the contact stress exerted onto shells
rwgaps	integer	flag for gap stiffness
rwgdth	real	death time for gap stiffness
rwksf	real	penalty scale factor
rwpnal	real	Scale factor for rigid wall penalties
sfric	real	Default static coefficient of friction
shledg	integer	Flag for assuming edge shape for shells
shlthk	integer	Shell thickness consideration flag
shltrw	real	Shell thickness scale factor
skiprwg	integer	Display rigidwall flag
slsfac	real	Scale factor for sliding penalties
spotdel	integer	Spotweld deletion flag
spothin	real	Optional thickness scale factor
spotstp	integer	Error termination flag on unfound spotweld
ssthk	integer	Shell thickness use flag for type 4 contacts
strobj	integer	Strong objectivity flag
swradf	real	Spot weld radius scale factor
tdcnof	integer	tied constraint offset contact update option
th	real	Default contact thickness
th_sf	real	Default thickness scale factor
thkchg	integer	Consider shell thickness change flag
tiedprj	integer	Projection bypass flag for TIED types
tieopt	integer	Option for constrained tied contact formulations
usrfrc	integer	Storage for user-controlled friction subroutine
usrstr	integer	Storage for user-controlled control subroutine

vfc	real	Default viscous friction coefficient
xpene	real	Surface max penetration check multiplier

*CONTROL_COUPLING

Properties for *CONTROL_COUPLING

Name	Type	Description
exists	logical	true if control card exists
flipx	integer	Flag to flip X coords
flipy	integer	Flag to flip Y coords
flipz	integer	Flag to flip Z coords
include	integer	The Include file number that the control card is in.
subcyl	integer	Subcycling flag
timidl	real	Idle time value
unforc	real	Force conversion factor
unleng	real	Length conversion factor
untime	real	Time conversion factor

*CONTROL_CPG

Properties for *CONTROL_CPG

Name	Type	Description
exists	logical	true if control card exists
ibchk	integer	Mesh integrity check
icorr	integer	Correction of density and total energy
include	integer	The Include file number that the control card is in.
iturb	integer	Turbulence model
ncpc	integer	Number of cycles between point cloud checks. A negative value will point to a load curve ID function of time
nslip	integer	Slip condition applied at walls
verb	integer	CPG verbosity control

*CONTROL_CPM

Properties for *CONTROL_CPM

Name	Type	Description
blkv	integer	Flag to allocate additional memory for contact nodal forces
cpmerr	integer	Disable/enable error checking
cpmmf	integer	Flag to consider airbag system velocity

exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
ncpmout	integer	Control CPM output database to D3PLOT
ncpmts	integer	Timestep size estimation
np2p	integer	Number of cycles for repartition particles
p2pmix	integer	Control the energy transfer during particle-to-particle collision
pmis	integer	Flag for choosing logic to use when a particle leaks out due to undetected contact
sffdc	real	Scale factor of force decay constant

*CONTROL_CPU

Properties for *CONTROL_CPU

Name	Type	Description
cputim	real	Max permitted cpu time
exists	logical	true if control card exists
iglst	integer	glstat data flag
include	integer	The Include file number that the control card is in.

*CONTROL_DEBUG

Properties for *CONTROL_DEBUG

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.

*CONTROL_DISCRETE_ELEMENT

Properties for *CONTROL_DISCRETE_ELEMENT

Name	Type	Description
ang	real	contact angle
bt	real	Birth time
cap	integer	dry/wet particle flag
cp	real	DES thermal property
dc	real	Exponential decay coefficient
dt	real	Death time
exists	logical	true if control card exists
fricd	real	Dynamic coefficient of friction
fricr	real	rolling friction coefficient

frics	real	friction coefficient
gamma	real	liquid surface tension
gap	real	parameter affecting spatial limit of liquid bridge
idesoft	integer	Flag for soft constraint formulation
ignore	integer	Ignore penetration flag
include	integer	The Include file number that the control card is in.
iskip	integer	Flag for skipping the calculation of contact force between DES:
lnorm	integer	LCID that defines the function for normal stiffness vs norm pen ratio
lshear	integer	LCID that defines the function for shear stiffness vs norm pen ratio
maxnei	integer	Number of neighbors to be tracked for DES contact and capillary force calculation
nbuf	integer	Asynchronous scheme and memory buffer option
ncrb	integer	Rebalancing frequency
ndamp	real	normal damping coefficient
normk	real	scale factor for normal spring constant
parallel	integer	Option to force calculation of bonded DES
sheark	real	ratio between sheark/normk
sofscl	real	Scale factor applied to the contact stiffness
tc	real	DES thermal property
tdamp	real	tangential damping coefficient
tfac	real	DES thermal property
vol	real	volume fraction
vtk	integer	max number of subcycling cycles

*CONTROL_DYNAMIC_RELAXATION

Properties for *CONTROL_DYNAMIC_RELAXATION

Name	**Type**	**Description**
drfctr	real	Dyn relaxation factor
drpset	integer	Part set used to check for convergence
drterm	real	Optional DR termination time
drtol	real	Convergence tolerance
edttl	real	Convergence tolerance on auto control
exists	logical	true if control card exists
idrflg	integer	Stress initialisation flag
include	integer	The Include file number that the control card is in.
irelal	integer	Automatic control flag
nrcyck	integer	#iterations between convergence checks
tssfdr	real	Optional timestep factor during DR

*CONTROL_EFG

Properties for *CONTROL_EFG

Name	Type	Description
etol	real	Error tolerance in the IMLM
exists	logical	true if control card exists
hsort	integer	Not used
ideb	integer	Output internal debug message
idila	integer	dilation param
imlm	integer	Choice for matrix operation, linear solving and memory usage
include	integer	The Include file number that the control card is in.
inint	integer	Factor needed for the estimation of maximum workspace used during initialization
ispline	integer	kernel function
ssort	integer	Flag for automatic sort of background triangular shells

*CONTROL_ENERGY

Properties for *CONTROL_ENERGY

Name	Type	Description
disen	integer	Dissipation energy calculation calc flag
drlen	integer	Drilling energy calculation flag
exists	logical	true if control card exists
hgen	integer	Hourglass energy calc flag
include	integer	The Include file number that the control card is in.
irgen	integer	Initial reference geometry calc flag
maten	integer	Detailed material energies flag
rwen	integer	Rigid wall energy calc flag
rylen	integer	Rayleigh energy calc flag
slnten	integer	Contact energy calc flag

*CONTROL_EOS_USER_LIBRARY

Properties for *CONTROL_EOS_USER_LIBRARY

Name	Type	Description
conl	real	Scaling factor for length conversion
conm	real	Scaling factor for mass conversion
conp	real	Scaling factor for pressure conversion
cont	real	Scaling factor for time conversion

exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
path	string	Path to library seslib
scle	real	Plot caling factor for energies
sclp	real	Plot caling factor for pressures
sclr	real	Plot caling factor for densities
sclt	real	Plot caling factor for temperatures

*CONTROL_EXPLICIT_THERMAL_ALE_COUPLING

Properties for *CONTROL_EXPLICIT_THERMAL_ALE_COUPLING

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
mmset	integer	The Multi-material Set ID.
partset	integer	The Part Set ID.

*CONTROL_EXPLICIT_THERMAL_BOUNDARY

Properties for *CONTROL_EXPLICIT_THERMAL_BOUNDARY

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
lcid	integer	The Curve ID specifying Temperature vs Time.
sgset	integer	The Segment Set ID.

*CONTROL_EXPLICIT_THERMAL_CONTACT

Properties for *CONTROL_EXPLICIT_THERMAL_CONTACT

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
ncycle	real	Number of cycle between checks of new contact.
partset	integer	The Part Set ID.

*CONTROL_EXPLICIT_THERMAL_INITIAL

Properties for *CONTROL_EXPLICIT_THERMAL_INITIAL

Name	Type	Description
exists	logical	true if control card exists
id	integer	If less than 0 then Element ID if greater than 0 then Set ID. Can be SOLID, SHELL, BEAM or THICK SHELL based on value of idtyp.
idtyp	integer	Type of ID. Valid values: 1-Solid, 2-Shell, 3-Beam, 4-Thick shell .
include	integer	The Include file number that the control card is in.
tempini	real	Initial Temperature.

*CONTROL_EXPLICIT_THERMAL_OUTPUT

Properties for *CONTROL_EXPLICIT_THERMAL_OUTPUT

Name	Type	Description
dtout	real/integer	Time interval between outputs. Constant float value if DTOUTYP = 0, Curve ID if DTOUTYP = 1.
dtoutyp	integer	Type of DTOUT. Valid values: 0-Constant, 1-Time vs DTOUT Curve.
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
set	integer	The Set ID. Can be SOLID, SHELL or BEAM Set based on value of setyp.
setyp	integer	Type of Set. Valid values: 1-Solid Set, 2-Shell Set, 3-Beam Set.

*CONTROL_EXPLICIT_THERMAL_PROPERTIES

Properties for *CONTROL_EXPLICIT_THERMAL_PROPERTIES

Name	Type	Description
cp	real/integer	Heat Capacity. Constant float value if CPTYP = 0, Curve ID if CPTYP = 1.
cptyp	integer	Type of CP. Valid values: 0-Constant, 1-Temperature vs Heat Capacity Curve.
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
kxx	real/integer	Heat conductivity matrix. Constant float value if respective KxxTYP = 0, Curve ID if respective KxxTYP = 1.
kxxtyp	integer	Types of Kxx. Valid values: 0-Constant, 1-Temperature vs Heat Conductivity Curve.
kxy	real/integer	Heat conductivity matrix. Constant float value if respective KxyTYP = 0, Curve ID if respective KxyTYP = 1.
kxytyp	integer	Types of Kxy. Valid values: 0-Constant, 1-Temperature vs Heat Conductivity Curve.
kxz	real/integer	Heat conductivity matrix. Constant float value if respective KxzTYP = 0, Curve ID if respective KxzTYP = 1.
kxztyp	integer	Types of Kxz. Valid values: 0-Constant, 1-Temperature vs Heat Conductivity Curve.
kyx	real/integer	Heat conductivity matrix. Constant float value if respective KyxTYP = 0, Curve ID if respective KyxTYP = 1.
kyxtyp	integer	Types of Kyx. Valid values: 0-Constant, 1-Temperature vs Heat Conductivity Curve.
kyy	real/integer	Heat conductivity matrix. Constant float value if respective KyyTYP = 0, Curve ID if respective KyyTYP = 1.

kyytyp	integer	Types of Kyy. Valid values: 0-Constant, 1-Temperature vs Heat Conductivity Curve.
kyz	real/integer	Heat conductivity matrix. Constant float value if respective KyzTYP = 0, Curve ID if respective KyzTYP = 1.
kyztyp	integer	Types of Kyz. Valid values: 0-Constant, 1-Temperature vs Heat Conductivity Curve.
kzx	real/integer	Heat conductivity matrix. Constant float value if respective KzxTYP = 0, Curve ID if respective KzxTYP = 1.
kzxtyp	integer	Types of Kzx. Valid values: 0-Constant, 1-Temperature vs Heat Conductivity Curve.
kzy	real/integer	Heat conductivity matrix. Constant float value if respective KzyTYP = 0, Curve ID if respective KzyTYP = 1.
kzytyp	integer	Types of Kzy. Valid values: 0-Constant, 1-Temperature vs Heat Conductivity Curve.
kzz	real/integer	Heat conductivity matrix. Constant float value if respective KzzTYP = 0, Curve ID if respective KzzTYP = 1.
kzztyp	integer	Types of Kzz. Valid values: 0-Constant, 1-Temperature vs Heat Conductivity Curve.
local	integer	Flag to activate an element csys. Valid values: 0-Vecids are considered in Global csys, 1-Vecids are considered in Local Csys.
partset	integer	The Part Set ID.
vecid1	integer	The Vector ID to define x-direction.
vecid2	integer	The Vector ID to define y-direction.

*CONTROL_EXPLICIT_THERMAL_SOLVER

Properties for *CONTROL_EXPLICIT_THERMAL_SOLVER

Name	Type	Description
dtfac	real	Time step factor.
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
partset	integer	The Part Set ID.

*CONTROL_EXPLOSIVE_SHADOW

Properties for *CONTROL_EXPLOSIVE_SHADOW

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
set_option	logical	true if _SET option is present.
setid	integer	Set ID of *SET_SHELL or *SET_SOLID.

*CONTROL_FORMING_BESTFIT

Properties for *CONTROL_FORMING_BESTFIT

Name	Type	Description

exists	logical	true if control card exists
filename	string	Target mesh file in keyword format
gaponly	integer	Separation distance calculation flag
ifast	integer	Computing performance optimisation flag
ifit	integer	Best fit flag
ifset	integer	Optional flag to define a node set to be included or excluded
include	integer	The Include file number that the control card is in.
nsets	integer	An optional node set ID of three nodes from the source mesh
nsett	integer	An optional node set ID of three nodes from the target mesh
nskip	integer	Optional skipping scheme
vector	logical	true if _VECTOR option is set

*CONTROL_FORMING_INITIAL_THICKNESS

Properties for *CONTROL_FORMING_INITIAL_THICKNESS

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
lcid	integer	Load curve ID defining thickness vs distance
pid	integer	Part ID of the sheet blank
vx	real	X component of vector defining the direction of distance in load curve
vy	real	Y component of vector defining the direction of distance in load curve
vz	real	Z component of vector defining the direction of distance in load curve
x0	real	Starting position x coordinate
y0	real	Starting position y coordinate
z0	real	Starting position z coordinate

*CONTROL_FORMING_MAXID

Properties for *CONTROL_FORMING_MAXID

Name	Type	Description
exists	logical	true if control card exists
i2dynain	integer	Keyword to be output to a dynain file
include	integer	The Include file number that the control card is in.
maxide	integer	Element ID number
maxidn	integer	Node ID number
pid	integer	Part ID of the sheet blank

*CONTROL_FORMING_POSITION

Properties for *CONTROL_FORMING_POSITION

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
pid	integer	Part ID
premove	real	Distance to pre-move tool in reverse direction
target	integer	

*CONTROL_FORMING_PRE_BENDING

Properties for *CONTROL_FORMING_PRE_BENDING

Name	Type	Description
cid	integer	ID of coordinate system (only for the LOCAL option)
cx	real	X component of centre of most-bent location
cy	real	Y component of centre of most-bent location
cz	real	Z component of centre of most-bent location
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
option	integer	Keyword option
pset	integer	Part set ID
radius	real	Radius of pre-bending
vx	real	X component of axis about which blank will be bent
vy	real	Y component of axis about which blank will be bent
vz	real	Z component of axis about which blank will be bent

*CONTROL_FORMING_PROJECTION

Properties for *CONTROL_FORMING_PROJECTION

Name	Type	Description
exists	logical	true if control card exists
gap	real	Minimum gap
include	integer	The Include file number that the control card is in.
nrbst	integer	Normal direction of blank
nrtst	integer	Normal direction of tool
pidb	integer	Part id for blank
pidt	integer	Part id for tool

*CONTROL_FORMING_REMOVE_ADAPTIVE_CONSTRAINTS

Properties for *CONTROL_FORMING_REMOVE_ADAPTIVE_CONSTRAINTS

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
pid	integer	Part id to remove adaptive constraints from

*CONTROL_FORMING_SHELL_TO_TSHELL

Properties for *CONTROL_FORMING_SHELL_TO_TSHELL

Name	Type	Description
exists	logical	true if control card exists
idsegb	integer	Set id of the segments to be generated at the bottom layer
idsegt	integer	Set id of the segments to be generated at the top layer
include	integer	The Include file number that the control card is in.
midsf	integer	Mid-plane position flag
pid	integer	Part id of the thin shell elements
thick	real	Thickness of the thick shell elements

*CONTROL_FORMING_STONING

Properties for *CONTROL_FORMING_STONING

Name	Type	Description
direct	real	Number of automatically determined directions
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
istone	integer	Stoning calculation option
itype	integer	Set type designation
length	real	Length of the stone
method	integer	Stoning method
node1	integer	Tail node defining stone moving direction
node1	integer	Head node defining stone moving direction
reverse	integer	Surface normal reversing option
sid	integer	Node/Shell set id
step	real	Stepping size of moving stone

v1	real	Vector component defining stoning direction
v2	real	Vector component defining stoning direction
v3	real	Vector component defining stoning direction
width	real	Width of the stone

*CONTROL_FORMING_STRAIN_RATIO_SMOOTH

Properties for *CONTROL_FORMING_STRAIN_RATIO_SMOOTH

Name	Type	Description
dt_cycle	real	Flag for output option (time interval or cycle number)
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
weight	real	Coefficient in equation

*CONTROL_FORMING_TEMPLATE

Properties for *CONTROL_FORMING_TEMPLATE

Name	Type	Description
al_fe	string	A=Aluminium blank, F=steel
amax	real	Maximum allowable acceleration
blkid	integer	Part (stype=0) or part set (stype=1) ID that defines the blank
bndl	integer	Part that defines the lower binder
bndu	integer	Part that defines the upper binder
d3plt	integer	Number of output states in the D3PLOT database
density	real	Density
dieid	integer	Part that defines the die
e	real	Youngs modulus
exists	logical	true if control card exists
fs	real	Friction coefficient
gap	real	Home gap between rigid tools
idtemp	integer	Type of forming process
include	integer	The Include file number that the control card is in.
k	real	Strength coefficient for exponential hardening
lcss	integer	Loadcurve for stress-strain relationship
lvlada	integer	Maximum adaptive level
mtyp	integer	Material type
n	real	Exponent for exponential hardening
patern	integer	Velocity profile of moving tool
pnch	integer	Part that defines the punch

pr	real	Poissons ratio
prebd	real	Distance between lower binder and punch
r00	real	Material anisotropic parameter R00
r45	real	Material anisotropic parameter R45
r90	real	Material anisotropic parameter R90
sizeada	real	Minimum element size permitted in the adaptive mesh
stype	integer	0->blkid is PART, 1->PARTSET NOTE don't use <type> as in stat_header
thick	real	Blank thickness
timsada	integer	Total number of adaptive steps during the forming simulation
unit	integer	Units for simulation
vid	integer	Vector ID defining direction of movement
vmax	real	Maximum allowable tool velocity
vx	real	X vector component of movement of punch
vy	real	Y vector component of movement of punch
vz	real	Z vector component of movement of punch

*CONTROL_FORMING_TOLERANC

Properties for *CONTROL_FORMING_TOLERANC

Name	Type	Description
dt_cycle	real	Flag for output option (time interval or cycle number)
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
output	integer	Output Flag
weight	real	Coefficient in equation

*CONTROL_FORMING_TRAVEL

Properties for *CONTROL_FORMING_TRAVEL

Name	Type	Description
exists	logical	true if control card exists
follow	integer	Part for tool to follow
gap	real	Min distance between tool and target in the home position
include	integer	The Include file number that the control card is in.
phase	integer	Phase number
pid	integer	Part ID of tool
target	integer	Move tool PID to meet part TARGET
travel	real	Distance to move tool along VID

vid	integer	Vector ID defining direction of travel

*CONTROL_FORMING_TRIM_MERGE

Properties for *CONTROL_FORMING_TRIM_MERGE

Name	Type	Description
exists	logical	true if control card exists
gapm	real	Gap distance between two open ends of a trim loop curve in the model
imerge	integer	Activation flag
include	integer	The Include file number that the control card is in.

*CONTROL_FORMING_TRIM_SOLID_REFINEMENT

Properties for *CONTROL_FORMING_TRIM_SOLID_REFINEMENT

Name	Type	Description
exists	logical	true if control card exists
ilevel	integer	Adaptive refinement level
include	integer	The Include file number that the control card is in.
irefine	integer	Flag to activate trimming of a multi-layer sandwiched part

*CONTROL_FORMING_UNFLANGING

Properties for *CONTROL_FORMING_UNFLANGING

Name	Type	Description
charlen	real	Max flange height
dist	real	Distance tolerance for auto-SPC along flange roots
dvid	integer	Not used
epsmx	real	Max effective plastic strain, beyond which elements are deleted
exists	logical	true if control card exists
iflimit	integer	Iteration limit for first phase of unfolding
ilinear	integer	Unfolding algorithm selection flag
include	integer	The Include file number that the control card is in.
nb1	integer	Start node ID on a flange root boundary
nb2	integer	ID of a node in the middl of the flange root boundary
nb3	integer	End node ID on a flange root boundary
ndouter	integer	A node ID on the outer flange boundary
noption	integer	Flag to turn on unfolding simulation
nunbend	integer	Estimated number of unbending

output	logical	TRUE if _<OPTION> is OUTPUT.
stfbend	real	Unflanging stiffness
stfcnt	real	Normal stiffness
thmn	real	Min thickness below which elements are deleted
thmx	real	Max thickness beyond which elements are deleted

*CONTROL_FORMING_USER

Properties for *CONTROL_FORMING_USER

Name	Type	Description
adatims	integer	Total number of adaptive steps during the forming simulation
al_fe	string	A=Aluminium blank, F=steel
amax	real	Maximum allowable acceleration
blank	integer	Part (stype=0) or part set (stype=1) ID for blank
d3plot	integer	Number of output states in the D3PLOT database
density	real	Density
e	real	Youngs modulus
exists	logical	true if control card exists
fs	real	Friction coefficient
gap	real	Minimum gap between tools
include	integer	The Include file number that the control card is in.
k	real	Strength coefficient for exponential hardening
lcss	integer	Loadcurve for stress-strain relationship
lvlada	integer	Maximum adaptive level
mtype	integer	Material type
n	real	Exponent for exponential hardening
patern	integer	Velocity profile of moving tool
pr	real	Poissons ratio
r00	real	Material anisotropic parameter R00
r45	real	Material anisotropic parameter R45
r90	real	Material anisotropic parameter R90
sizeada	real	Minimum element size permitted in the adaptive mesh
stype	integer	Flag for part/part set
thick	real	Blank thickness
unit	integer	Units for simulation
vmax	real	Maximum allowable tool velocity

*CONTROL_FREQUENCY_DOMAIN

Properties for *CONTROL_FREQUENCY_DOMAIN

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
mcf	integer	Flag for writing out MCF (Modal Coefficient File) from SSD analysis
mpn	real	Large mass added per node.
refgeo	integer	Flag for reference geometry in acoustic eigenvalue analysis

*CONTROL_FREQUENCY_RESPONSE_FUNCTION

Properties for *CONTROL_FREQUENCY_RESPONSE_FUNCTION

Name	Type	Description
dampf	real	Modal damping coefficient
dmpmas	real	Mass proportional damping constant in Rayleigh damping
dmpstf	real	Stiffness proportional damping constant in Rayleigh damping
dof1	integer	Applicable degrees-of-freedom for excitation input
dof2	integer	Applicable degrees-of-freedom for response output
exists	logical	true if control card exists
fmax	real	Maximum frequency for FRF output
fmin	real	Minimum frequency for FRF output
fnmax	real	Optional maximum natural frequency
include	integer	The Include file number that the control card is in.
lcdam	integer	Loadcurve ID defining modal damping coefficient
lctyp	integer	Type of load curve
mdmax	integer	Last mode employed in FRF computation
mdmin	integer	First mode employed in FRF computation
n1	integer	Node (n1typ=0) / node set (n1typ=1) /segment set (n1typ=2) ID for excitation input
n1typ	integer	Type of N1
n2	integer	Node (n2typ=0) /node set (n2typ=1) /segment set (n2typ=2) ID for response output
n2typ	integer	Type of N2
nfreq	integer	Number of frequencies for FRF output
restrt	integer	Restart option
vad1	integer	Excitation input type
vad2	integer	Response output type
vid	integer	Vector ID for DOF1=4

*CONTROL_HOURGLASS

Properties for *CONTROL_HOURGLASS

Name	Type	Description
exists	logical	true if control card exists
f_936	integer	Internal flag to set 936 compatibility
ihq	integer	Hourglass viscosity type
include	integer	The Include file number that the control card is in.
qh	real	Default hourglass coefficient

*CONTROL_IMPLICIT_AUTO

Properties for *CONTROL_IMPLICIT_AUTO

Name	Type	Description
dtexp	real	time in explicit before switch
dtmax	integer	Maximum allowable timestep. Loadcurve if negative
dtmin	real	Minimum allowable timestep
exists	logical	true if control card exists
hcmax	integer	Mid-point relative Euclidian residual norm max tolerance
hcmin	integer	Mid-point relative Euclidian residual norm min tolerance
hmmax	integer	Mid-point relative maximum residual norm max tolerance
hmmin	integer	Mid-point relative maximum residual norm min tolerance
hnrmax	integer	Mid-point absolute Nodal Rotational norm tolerance
hntmax	integer	Mid-point absolute Nodal Translational norm tolerance
hrrmax	integer	Mid-point absolute Rigid body Rotational norm tolerance
hrtmax	integer	Mid-point absolute Rigid body Translational norm tolerance
iauto	integer	Automatic timestep control flag. Loadcurve if negative
include	integer	The Include file number that the control card is in.
iteopt	integer	Optimum equilibrium iteration count per timestep
itewin	integer	Allowable iteration window (no. of iterations)
kcycle	integer	number of explicit cycles before switch
kfail	integer	number of failed implicit attempts before switch

*CONTROL_IMPLICIT_BUCKLE

Properties for *CONTROL_IMPLICIT_BUCKLE

Name	Type	Description
bckmth	integer	Method to extract buckling modes
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
nmode	integer	number of buckling modes to calculate

*CONTROL_IMPLICIT_CONSISTENT_MASS

Properties for *CONTROL_IMPLICIT_CONSISTENT_MASS

Name	Type	Description
exists	logical	true if control card exists
iflag	integer	Consistent mass matrix flag
include	integer	The Include file number that the control card is in.

*CONTROL_IMPLICIT_DYNAMICS

Properties for *CONTROL_IMPLICIT_DYNAMICS

Name	Type	Description
alpha	real	Composite time integration constant
beta	real	Newmark time integration constant
exists	logical	true if control card exists
gamma	real	Newmark time integration constant
imass	integer	Implicit analysis type
include	integer	The Include file number that the control card is in.
irate	integer	rate effect switch
tdybir	integer	birth time for dynamic terms. Loadcurve if negative
tdybur	real	burial
tdydth	real	death

*CONTROL_IMPLICIT_EIGENVALUE

Properties for *CONTROL_IMPLICIT_EIGENVALUE

Name	Type	Description
center	real	Centre frequency
eigmscl	integer	Control whether to use the original (unscaled) or the scaled mass
eigmth	integer	Eigenvalue extraction method
evdump	integer	Flag for writing eigenvalues and eigenvectors
exists	logical	true if control card exists
ibeam	integer	Beam element formulation for implicit
include	integer	The Include file number that the control card is in.
iparm1	integer	Minimum block size for the Cholesky factorization (for eigmth=101) or Maximum number of iterations (for eigmth=102)
iparm2	integer	Maximum block size for the Cholesky factorization (for eigmth=101) or Block size (for eigmth=102)
iparm3	integer	Node set ID

iparm4	integer	MCMS minimum group/substructure size
iparm5	integer	Harmonic index
iparm6	integer	Vector ID for the axis of rotation
ishell	integer	Shell element formulation for implicit
isolid	integer	Solid element formulation for implicit
itshell	integer	Thick shell element formulation for implicit
lflag	integer	Left end point finite flag
lftend	real	Left end point of interval
mstres	integer	stress compute flag
mstrscl	real	Scaling for computing velocity
neig	integer	#eigenvalues to extract; loadcurve if negative
rflag	integer	Right end point finite flag
rhtend	real	Right end point of interval
rotscl	real	Scale factor for the inertia of rotational degrees of freedom
rparm1	real	Eigenvalue expansion factor (for eigmth=101) or Convergence tolerance (for eigmth=102)
rparm2	real	BLR preconditioner tolerance
rparm4	real	Control output of eigenvectors to the d3eigv database
shfscl	real	Shift scale

*CONTROL_IMPLICIT_EXPLICIT_HYBRID

Properties for *CONTROL_IMPLICIT_EXPLICIT_HYBRID

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
psid	integer	Part set ID

*CONTROL_IMPLICIT_FORMING

Properties for *CONTROL_IMPLICIT_FORMING

Name	Type	Description
birth	real	birth time
death	real	death time
dt0	real	initial time step size
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
ioption	integer	1:gravity 2:binder
nsmax	integer	max number of implicit steps
nsmin	integer	min number of implicit steps

penchk	real	penetration allowed as ratio of part thickness

*CONTROL_IMPLICIT_GENERAL

Properties for *CONTROL_IMPLICIT_GENERAL

Name	Type	Description
cnstn	integer	Consistent tangent stiffness flag
dt0	real	Initial timestep for implicit analysis
exists	logical	true if control card exists
form	integer	Element formulation to use.
igs	integer	Geometric (initial stress) stiffness flag
imflag	integer	Implicit/explicit switching flag; loadcurve if negative
imform	integer	Element formulation switching flag
include	integer	The Include file number that the control card is in.
nsbs	integer	Number of steps in non-linear springback
zero_v	integer	flag to zero vels before switch to implicit

*CONTROL_IMPLICIT_INERTIA_RELIEF

Properties for *CONTROL_IMPLICIT_INERTIA_RELIEF

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
ircnt	integer	Lowest IRCNT modes
irflag	integer	Inertia relief flag
mxmodes	integer	Maximum number of modes stored in memory
thresh	real	Threshold for rigid body node

*CONTROL_IMPLICIT_JOINTS

Properties for *CONTROL_IMPLICIT_JOINTS

Name	Type	Description
exists	logical	true if control card exists
icylin	integer	Treatment of cylindrical joints
include	integer	The Include file number that the control card is in.
irevol	integer	Treatment of revolute joints
ispher	integer	Treatment of spherical joints

*CONTROL_IMPLICIT_MODAL_DYNAMIC

Properties for *CONTROL_IMPLICIT_MODAL_DYNAMIC

Name	Type	Description
dtout	real	Modal dynamics output interval
exists	logical	true if control card exists
filename2	string	Constraint modes file name
filename3	string	Residual vectors file name
filename	string	Eigen modes file name
include	integer	The Include file number that the control card is in.
integ	integer	Integration method
md_strs	integer	Modal dynamic stress flag
mdflag	integer	Modal dynamic flag
nsid	integer	Node set ID of the nodes in the modal model that are subjected to loads
zeta	real	Modal dynamic damping constant

*CONTROL_IMPLICIT_MODAL_DYNAMIC_DAMPING

Member functions

- GetCoefficient(index*[integer]*)
- RemoveCoefficient(index*[integer]*)
- SetCoefficient(index*[integer]*, mode/frequency*[integer/real]*, zeta*[real]*)

Properties for *CONTROL_IMPLICIT_MODAL_DYNAMIC_DAMPING

Name	Type	Description
coefficients (read only)	integer	Number of coefficients for SPECIFIC and FREQUENCY_RANGE options
exists	logical	true if control card exists
frequency_range	boolean	If FREQUENCY_RANGE option is used
include	integer	The Include file number that the control card is in.
specific	boolean	If SPECIFIC option is used
zeta	real	Modal dynamic damping constant

Details of functions

GetCoefficient(index*[integer]*)

Description

Returns the damping coefficient data for an index in *CONTROL_IMPLICIT_MODAL_DYNAMIC_DAMPING.

Arguments

- **index** (integer)

The index you want the data for. **Note that indices start at 0, not 1**.

Returns

An array containing the mode id/frequency and damping coefficient values.

Return type

Array

Example

To get the damping data for the 3rd index for *CONTROL_IMPLICIT_MODAL_DYNAMIC_DAMPING in model m:

```
if (m.control.implicit_modal_dynamic_damping.coefficients >= 3)
{
    var data = m.control.implicit_modal_dynamic_damping.GetCoefficient(2);
}
```

RemoveCoefficient(index*[integer]*)

Description

Removes the damping coefficient data for an index in *CONTROL_IMPLICIT_MODAL_DYNAMIC_DAMPING.

Arguments

- **index** (integer)

The index you want to delete damping data for. **Note that indices start at 0, not 1**.

Returns

No return value.

Example

To delete the damping data for the 3rd index for *CONTROL_IMPLICIT_MODAL_DYNAMIC_DAMPING in model m:

```
if (m.control.implicit_modal_dynamic_damping.coefficients >= 3)
{
    m.control.implicit_modal_dynamic_damping.RemoveCoefficient(2);
}
```

SetCoefficient(index*[integer]*, mode/frequency*[integer/real]*, zeta*[real]*)

Description

Sets the damping coefficient data for an index in *CONTROL_IMPLICIT_MODAL_DYNAMIC_DAMPING.

Arguments

- **index** (integer)

The index you want to set the data for. **Note that indices start at 0, not 1**.

- **mode/frequency** (integer/real)

The mode ID (_SPECIFIC) or frequency (_FREQUENCY_RANGE).

- **zeta** (real)

Damping coefficient

Returns

No return value.

Example

To set the damping data for the 3rd index for *CONTROL_IMPLICIT_MODAL_DYNAMIC_DAMPING_SPECIFIC in model m to have mode ID 10 and damping coefficient 0.1:

```
m.control.implicit_modal_dynamic_damping.SetCoefficient(2, 10, 0.1);
```

*CONTROL_IMPLICIT_MODES

Properties for *CONTROL_IMPLICIT_MODES

Name	Type	Description
exists	logical	true if control card exists
ibase	integer	Offset for numbering
id3mode	integer	Write d3mode file flag
include	integer	The Include file number that the control card is in.
iresvec	integer	Converting the attachment modes to residual vectors flag
istress	integer	Flag to compute stresses
neig	integer	Number of eigenmodes
nsida	integer	node set for attachment modes
nsidc	integer	node set constraint modes
opt	integer	Can be <BLANK> or _BINARY
se_damp	string	Name of superelement damping matrix
se_filename	string	File name
se_inert	string	Name of superelement inertia matrix
se_mass	string	Name of superelement mass matrix
se_stiff	string	Name of superelement stiffness matrix

*CONTROL_IMPLICIT_ORDERING

Properties for *CONTROL_IMPLICIT_ORDERING

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
nmetis	integer	Number of times to use Metis
order	integer	Ordering option

*CONTROL_IMPLICIT_RESIDUAL_VECTOR

Properties for *CONTROL_IMPLICIT_RESIDUAL_VECTOR

Name	Type	Description
exists	logical	true if control card exists
iformat	integer	Format for processing eigenmodes
include	integer	The Include file number that the control card is in.
iresvec	integer	Residual vector control flag
neig	integer	Number of eigenmodes to compute for the purpose of orthogonalizing the computed load

*CONTROL_IMPLICIT_SOLUTION

Properties for *CONTROL_IMPLICIT_SOLUTION

Name	Type	Description
abstol	real	absolute convergence tol
arcalf	integer	relative influence predictor step
arcctl	integer	Arc length controlling node ID
arcdir	integer	Arc length controlling node direction
arcdmp	integer	Arc length damping option
arclen	real	Arc length size
arcmth	integer	Arc length method
arcpsi	integer	relative influence load/time parameter
arctim	integer	initiation time
awgt	real	weight factor
cpchk	integer	Contact penetration check flag
d3itctl	integer	D3ITER database control
dctol	real	Displacement convergence tolerance
diverg	integer	Divergence flag
dmtol	real	Maximum displacement convergence tolerance
dnorm	integer	Displacement norm for convergence test
ectol	real	Energy convergence tolerance
emtol	real	Maximum energy convergence tolerance
exists	logical	true if control card exists
ilimit	integer	Iteration limit between automatic stiffness reformations
include	integer	The Include file number that the control card is in.
irad	real	curve factor
istif	integer	Initial stiffness formulation flag
kssize	integer	size of krylov space
lsdir	integer	search direction
lsmtd	integer	search method
lstol	real	Line search convergence tolerance
maxref	integer	Stiffness reformation limit per time step

nlnorm	real	non-linear convergence type
nlprint	integer	non-linear solver print flag
nrtol	real	Nodal rotational convergence tolerance
nsolvr	integer	Non-linear equation solver method
nttol	real	Nodal translational convergence tolerance
rctol	real	Residual (force) convergence tolerance
rmtol	real	Maximum residual convergence tolerance
rrtol	real	Rigid body rotational convergence tolerance
rttol	real	Rigid body translational convergence tolerance
srad	real	radius of influence
sred	real	step reduction factor

*CONTROL_IMPLICIT_SOLVER

Properties for *CONTROL_IMPLICIT_SOLVER

Name	Type	Description
absmem	string	Absolute upper bound for the dynamic memory allocated for factorization
autospc	integer	AUTOSPC switch
autotol	real	AUTOSPC tolerance
drcm	integer	Drilling rotation constraint method
drcprm	real	Drilling rotation constraint parameter
emxdmp	integer	Flag for dumping elemental stiffness and mass matrices
exists	logical	true if control card exists
iblropt	integer	Flag providing options for the block low-rank (BLR) computations
include	integer	The Include file number that the control card is in.
iparm1	integer	Maximum number of iterations
isingle	integer	Controls tasks performed in single precision
ispd	integer	Symmetric positive definite flag
lcpack	integer	Matrix assembly package
lprint	integer	Linear solver print flag
lsolvr	integer	Linear equation solver method
memlvl	real	Control the load-balancing behavior
mtxdmp	integer	flag to dump matrix
negev	integer	Negative eigenvalue flag
order	integer	Ordering option
rdcmem	integer	Factor for capping the amount of dynamic memory requested
rparm1	integer	Absolute tolerance for convergence
rparm2	integer	Relative tolerance for convergence
rparm4	integer	Nominal block size

rparm5	integer	Compression tolerance for the low-rank approximation

*CONTROL_IMPLICIT_SSD_DIRECT

Properties for *CONTROL_IMPLICIT_SSD_DIRECT

Name	Type	Description
exists	logical	true if control card exists
fmax	real	Maximum frequency in the solution
fmin	real	Minimum frequency in the solution
fractn	integer	Octave fraction
fspace	real	Solution frequency assignment strategy
include	integer	The Include file number that the control card is in.
issflg	integer	Steady state vibration flag
loss	real	Structural loss factor
nfreq	integer	Number of frequencies in the solution

*CONTROL_IMPLICIT_STABILIZATION

Properties for *CONTROL_IMPLICIT_STABILIZATION

Name	Type	Description
exists	logical	true if control card exists
ias	integer	Artificial stabilization flag
include	integer	The Include file number that the control card is in.
scale	integer	scale factor for artificial stabilization. Loadcurve if negative
tend	real	End time
tstart	real	Start time

*CONTROL_IMPLICIT_STATIC_CONDENSATION

Properties for *CONTROL_IMPLICIT_STATIC_CONDENSATION

Name	Type	Description
binary	integer	flag to set _BINARY option
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
sc_flag	integer	Static condensation control flag
sc_nsid	integer	Node set ID for nodes to be preserved in the procedure
sc_psid	integer	Part set ID for parts to be included in the procedure
se_filename	string	File name

se_inert	string	Name of superelement inertia matrix
se_mass	string	Name of superelement mass matrix
se_stiff	string	Name of superelement stiffness matrix

*CONTROL_IMPLICIT_TERMINATION

Properties for *CONTROL_IMPLICIT_TERMINATION

Name	Type	Description
absol	real	Terminate based on absolute total displacement in the Euclidean norm.
delta1	real	Terminate based on rel total displacement in max norm
deltau	real	Terminate based on rel total displacement in Euclidean norm
exists	logical	true if control card exists
ietol	real	Terminate based on internal energy
include	integer	The Include file number that the control card is in.
ketol	real	Terminate based on kinetic energy
nstep	integer	Consecutive implicit time steps
tetol	real	Terminate based on total energy

*CONTROL_MAPPING_SHIFT_BY_DT

Properties for *CONTROL_MAPPING_SHIFT_BY_DT

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.

*CONTROL_MPP_CONTACT_GROUPABLE

Properties for *CONTROL_MPP_CONTACT_GROUPABLE

Name	Type	Description
exists	logical	true if control card exists
grp	integer	GROUPABLE algorithm options
include	integer	The Include file number that the control card is in.

*CONTROL_MPP_DECOMPOSITION_ADJUST_MTYPE_COST

Member functions

- GetTypeCost(index*[integer]*)
- RemoveCost(index*[integer]*)
- SetTypeCost(index*[integer]*, Type Id *[integer]*, A *[real]*, B*[real]*)

Properties for *CONTROL_MPP_DECOMPOSITION_ADJUST_MTYPE_COST

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
num_type_costs (read only)	integer	Number of Material type costs

Details of functions

GetTypeCost(index*[integer]*)

Description

Returns the cost data for an index in *CONTROL_MPP_DECOMPOSITION_ADJUST_MTYPE_COST.

Arguments

- **index** (integer)

The index you want the cost data for. **Note that indices start at 0, not 1**.

Returns

An array containing the typeid, a and b values.

Return type

Array

Example

To get the cost data for the 3rd index for *CONTROL_MPP_DECOMPOSITION_ADJUST_MTYPE_COST in model m:

```
if (m.control.mpp_decomposition_adjust_mtype_cost.num_type_costs >= 3)
{
    var data = m.control.mpp_decomposition_adjust_mtype_cost.GetTypeCost(2);
}
```

RemoveCost(index*[integer]*)

Description

Removes the cost data for an index in *CONTROL_MPP_DECOMPOSITION_ADJUST_MTYPE_COST.

Arguments

- **index** (integer)

The index you want to delete the cost data for. **Note that indices start at 0, not 1**.

Returns

No return value.

Example

To delete the cost data for the 3rd index for *CONTROL_MPP_DECOMPOSITION_ADJUST_MTYPE_COST in model m:

```
if (m.control.mpp_decomposition_adjust_mtype_cost.num_type_costs >= 3)
{
    m.control.mpp_decomposition_adjust_mtype_cost.RemoveTypeCost(2);
}
```

SetTypeCost(index*[integer]*, Type Id *[integer]*, A *[real]*, B*[real]*)

Description

Sets the cost data for an index in *CONTROL_MPP_DECOMPOSITION_ADJUST_MTYPE_COST.

Arguments

- **index** (integer)

The index you want to set the data for. **Note that indices start at 0, not 1**.

- **Type Id** (integer)

Material type as a number.

- **A** (real)

Cost Mutlipier.

- **B** (real)

Added Cost

Returns

No return value.

Example

To set the data for the 3rd index for *CONTROL_MPP_DECOMPOSITION_ADJUST_MTYPE_COST in model m to have Mat Type 10 , Cost Multlier 5 and Added cost 6:

```
m.control.mpp_decomposition_adjust_mtype_cost.SetTypeCost(2, 10, 5.0 , 6.0);
```

---

*CONTROL_MPP_DECOMPOSITION_ADJUST_PART_COST

Member functions

- GetPartCost(index*[integer]*)
- RemovePartCost(index*[integer]*)
- SetPartCost(index*[integer]*, PID*[integer]*, A*[real]*, B*[real]*)

Properties for *CONTROL_MPP_DECOMPOSITION_ADJUST_PART_COST

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
num_part_costs (read only)	integer	Number of part costs.

Details of functions

GetPartCost(index*[integer]*)

Description

Returns the part cost data for an index in *CONTROL_MPP_DECOMPOSITION_ADJUST_PART_COST.

Arguments

- **index** (integer)

The index you want the data for. **Note that indices start at 0, not 1**.

Returns

An array containing the typeid, a and b values.

Return type

Array

Example

To get the cost data for the 3rd index for *CONTROL_MPP_DECOMPOSITION_ADJUST_PART_COST in model m:

```
if (m.control.mpp_decomposition_adjust_part_cost.num_part_costs >= 3)
{
    var data = m.control.mpp_decomposition_adjust_part_cost.GetPartCost(2);
}
```

RemovePartCost(index*[integer]*)

Description

Removes the part cost data for an index in *CONTROL_MPP_DECOMPOSITION_ADJUST_PART_COST.

Arguments

- **index** (integer)

The index you want to part cost data for. **Note that indices start at 0, not 1**.

Returns

No return value.

Example

To delete the part cost data for the 3rd index for *CONTROL_MPP_DECOMPOSITION_ADJUST_PART_COST in model m:

```
if (m.control.mpp_decomposition_adjust_part_cost.num_part_costs >= 3)
{
    m.control.mpp_decomposition_adjust_part_cost.RemovePartCost(2);
}
```

SetPartCost(index*[integer]*, PID*[integer]*, A*[real]*, B*[real]*)

Description

Sets the part cost data for an index in *CONTROL_MPP_DECOMPOSITION_ADJUST_PART_COST.

Arguments

- **index** (integer)

The index you want to set the data for. **Note that indices start at 0, not 1**.

- **PID** (integer)

Part Id.

- **A** (real)

Cost Mutliplier

- **B** (real)

Added Cost.

Returns

No return value.

Example

To set the cost data for the 3rd index for *CONTROL_MPP_DECOMPOSITION_ADJUST_PART_COST in model m to have Part 7 , Cost Multiplier 0.1 and Added Cost 5.2:

```
m.control.mpp_decomposition_adjust_part_cost.SetPartCost(2, 7, 0.1, 5.2);
```

*CONTROL_MPP_DECOMPOSITION_AUTOMATIC

Properties for *CONTROL_MPP_DECOMPOSITION_AUTOMATIC

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.

*CONTROL_MPP_DECOMPOSITION_BAGREF

Properties for *CONTROL_MPP_DECOMPOSITION_BAGREF

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.

*CONTROL_MPP_DECOMPOSITION_CHECK_SPEED

Properties for *CONTROL_MPP_DECOMPOSITION_CHECK_SPEED

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.

*CONTROL_MPP_DECOMPOSITION_CONTACT_ISOLATE

Properties for *CONTROL_MPP_DECOMPOSITION_CONTACT_ISOLATE

Name	Type	Description
exists	logical	true if control card exists
id1	integer	Contact ID 1 to distrubute
id2	integer	Contact ID 2 to distrubute
id3	integer	Contact ID 3 to distrubute
id4	integer	Contact ID 4 to distrubute
id5	integer	Contact ID 5 to distrubute

include	integer	The Include file number that the control card is in.

*CONTROL_MPP_DECOMPOSITION_DISABLE_UNREF_CURVES

Properties for *CONTROL_MPP_DECOMPOSITION_DISABLE_UNREF_CURVES

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.

*CONTROL_MPP_DECOMPOSITION_DISTRIBUTE_ALE_ELEMENTS

Properties for *CONTROL_MPP_DECOMPOSITION_DISTRIBUTE_ALE_ELEMENTS

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
overlap	logical	Decompose the structure and ALE domains together?

*CONTROL_MPP_DECOMPOSITION_DISTRIBUTE_SPH_ELEMENTS

Properties for *CONTROL_MPP_DECOMPOSITION_DISTRIBUTE_SPH_ELEMENTS

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.

*CONTROL_MPP_DECOMPOSITION_ELCOST

Properties for *CONTROL_MPP_DECOMPOSITION_ELCOST

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
itype	integer	Hardware specific cost profile

*CONTROL_MPP_DECOMPOSITION_FILE

Properties for *CONTROL_MPP_DECOMPOSITION_FILE

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
name	string	decomposition file

*CONTROL_MPP_DECOMPOSITION_FLAG_STRESS_STRAIN_CURVE

Properties for *CONTROL_MPP_DECOMPOSITION_FLAG_STRESS_STRAIN_CURVE

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.

*CONTROL_MPP_DECOMPOSITION_METHOD

Properties for *CONTROL_MPP_DECOMPOSITION_METHOD

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
name	string	decomposition method

*CONTROL_MPP_DECOMPOSITION_NUMPROC

Properties for *CONTROL_MPP_DECOMPOSITION_NUMPROC

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
n	integer	number of processors

*CONTROL_MPP_DECOMPOSITION_OUTDECOMP

Properties for *CONTROL_MPP_DECOMPOSITION_OUTDECOMP

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.

itype	integer	Database format

*CONTROL_MPP_DECOMPOSITION_RCBLOG

Properties for *CONTROL_MPP_DECOMPOSITION_RCBLOG

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
name	string	decomposition file

*CONTROL_MPP_DECOMPOSITION_ REDECOMPOSITION

Properties for *CONTROL_MPP_DECOMPOSITION_REDECOMPOSITION

Name	Type	Description
defgeo	integer	Geometry for decomposition
exists	logical	true if control card exists
freq	real	Time interval between redecomposition
include	integer	The Include file number that the control card is in.
remsph	integer	Flag to remove deactivated SPH particles
sampt	real	Time interval for collecting element cost profile to use in the next REDECOMP step.
stime	real	Start time for redecomposition
weight	real	Element cost scale factor for element in contact

*CONTROL_MPP_DECOMPOSITION_SCALE_ CONTACT_COST

Properties for *CONTROL_MPP_DECOMPOSITION_SCALE_CONTACT_ COST

Name	Type	Description
exists	logical	true if control card exists
id10	integer	Contact ID 10 to distrubute
id11	integer	Contact ID 11 to distrubute
id12	integer	Contact ID 12 to distrubute
id13	integer	Contact ID 13 to distrubute
id14	integer	Contact ID 14 to distrubute
id15	integer	Contact ID 15 to distrubute
id1	integer	Contact ID 1 to distrubute

id2	integer	Contact ID 2 to distrubute
id3	integer	Contact ID 3 to distrubute
id4	integer	Contact ID 4 to distrubute
id5	integer	Contact ID 5 to distrubute
id6	integer	Contact ID 6 to distrubute
id7	integer	Contact ID 7 to distrubute
id8	integer	Contact ID 8 to distrubute
id9	integer	Contact ID 9 to distrubute
include	integer	The Include file number that the control card is in.
sf	real	Scale factor

*CONTROL_MPP_DECOMPOSITION_SCALE_FACTOR_SPH

Properties for *CONTROL_MPP_DECOMPOSITION_SCALE_FACTOR_SPH

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
sf	real	Scale factor

*CONTROL_MPP_DECOMPOSITION_SHOW

Properties for *CONTROL_MPP_DECOMPOSITION_SHOW

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.

*CONTROL_MPP_DECOMPOSITION_TRANSFORMATION

Member functions

- GetTransformation(row*[integer]*)
- RemoveTransformation(row*[integer]*)
- SetTransformation(row*[integer]*, type or array*[string or array]*, V1*[real]*, V2 to V9 (optional)*[real]*)

Properties for *CONTROL_MPP_DECOMPOSITION_TRANSFORMATION

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.

rows	integer	Number of rows.

Details of functions

GetTransformation(row*[integer]*)

Description

Returns the transformation for a row in *CONTROL_MPP_DECOMPOSITION_TRANSFORMATION.

Arguments

- **row** (integer)

The row you want the data for. **Note that rows start at 0, not 1**.

Returns

An array containing the transformation type and the parameters V1 to V9.

Return type

Array

Example

To get the transformation for the 3rd row for *CONTROL_MPP_DECOMPOSITION_TRANSFORMATION in model m:

```
if(m.control.mpp_decomposition_transformation.rows >= 3)
{
    var type = m.control.mpp_decomposition_transformation.GetTransformation(2);
}
```

RemoveTransformation(row*[integer]*)

Description

Removes a row in *CONTROL_MPP_DECOMPOSITION_TRANSFORMATION.

Arguments

- **row** (integer)

The row number to be deleted. **Note that rows start at 0, not 1**. If there are rows under this one, they will be shifted up.

Returns

No return value.

Example

To delete the 3rd row for *CONTROL_MPP_DECOMPOSITION_TRANSFORMATION in model m:

```
if(m.control.mpp_decomposition_transformation.rows >= 3)
{
    m.control.mpp_decomposition_transformation.RemoveTransformation(2);
}
```

SetTransformation(row*[integer]*, type or array*[string or array]*, V1*[real]*, V2 to V9 (optional)*[real]*)

Description

Sets the type and the parameters V1 to V9 for a row in *CONTROL_MPP_DECOMPOSITION_ TRANSFORMATION. Note: If the row already exists, the type and the parameters will be overwritten.

Arguments

- **row** (integer)

The row you want to set the data for. **Note that rows start at 0, not 1**.

- **type or array** (string or array)

The string repesenting the type of the transformation. Alternatively an array with this and the following parameters on it.

- **V1** (real)

The float repesenting the parameter V1

- **V2 to V9 (optional)** (real)

The floats repesenting the parameters V2 to V9. These are only used for VEC3, C2R, S2R and MAT.

Returns

No return value.

Example

To set in the first row a transformation of type to RZ with V1=1.2 for *CONTROL_MPP_DECOMPOSITION_TRANSFORMATION in model m:

```
m.control.implicit_modal_dynamic_damping.SetTransformation(0, "RZ", 1.2);
```

or

```
var a = new Array("RZ", 1.2);
      m.control.implicit_modal_dynamic_damping.SetTransformation(0, a);
```

To set in the third row a transformation of type to VEC3 with V1=1.0, V2=2.0... V9=9.0 for *CONTROL_MPP_DECOMPOSITION_TRANSFORMATION in model m:

```
m.control.implicit_modal_dynamic_damping.SetTransformation(2, "VEC3", 1, 2, 3,
4, 5, 6, 7, 8, 9);
```

or

```
var a = new Array("VEC3", 1, 2, 3, 4, 5, 6, 7, 8, 9);
      m.control.implicit_modal_dynamic_damping.SetTransformation(2, a);
```

*CONTROL_MPP_IO_BINOUTONLY

Properties for *CONTROL_MPP_IO_BINOUTONLY

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.

*CONTROL_MPP_IO_LSTC_REDUCE

Properties for *CONTROL_MPP_IO_LSTC_REDUCE

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.

*CONTROL_MPP_IO_NOD3DUMP

Properties for *CONTROL_MPP_IO_NOD3DUMP

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.

*CONTROL_MPP_IO_NODUMP

Properties for *CONTROL_MPP_IO_NODUMP

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.

*CONTROL_MPP_IO_NOFAIL

Properties for *CONTROL_MPP_IO_NOFAIL

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.

*CONTROL_MPP_IO_NOFULL

Properties for *CONTROL_MPP_IO_NOFULL

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.

*CONTROL_MPP_IO_SWAPBYTES

Properties for *CONTROL_MPP_IO_SWAPBYTES

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.

*CONTROL_MPP_MATERIAL_MODEL_DRIVER

Properties for *CONTROL_MPP_MATERIAL_MODEL_DRIVER

Name	Type	Description

exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.

*CONTROL_MPP_REBALANCE

Properties for *CONTROL_MPP_REBALANCE

Name	Type	Description
exists	logical	true if control card exists
icoor	integer	Coordinates used in rebalance
icost	integer	Element costs used in rebalance
include	integer	The Include file number that the control card is in.
ncycle	integer	Number of cycles between rebalance steps
thres	real	Percent threshold for rebalancing

*CONTROL_NONLOCAL

Properties for *CONTROL_NONLOCAL

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
mem	integer	%age increase in memory for *MAT_NONLOCAL usage

*CONTROL_OUTPUT

Properties for *CONTROL_OUTPUT

Name	Type	Description
cdetol	real	Tolerance for output of *DEFINE_CURVE discretization warnings
demden	integer	Output DEM density data to D3PLOT database
engout	integer	Flag to output contact sliding energy densities for mortar contact
eocs	integer	Elout coordinate system option
exists	logical	true if control card exists
frfreq	integer	Output frequency for failed element report
gmdt	real	output interval for *INTERFACE_SSI_AUX
hisnout	integer	Flag to invoke output of extra history variable names
iaccop	integer	Flag for accels in d3thdt to be averaged
ibsf	integer	Flag to invoke output of *SET_BEAM data
icrfile	integer	Output node and element sets used in computing secforc data
ielogini	integer	Flag to invoke diagnostic output to error file during the initialisation phase

ielogkey	integer	Flag to invoke diagnostic output to error file during the input phase
ielogsol	integer	Flag to invoke diagnostic output to error file during the solution phase
ierode	integer	output eroded energy
iflush	integer	i/o buffer flushing interval (t-steps)
ikedit	integer	Status report interval to d3hsp
include	integer	The Include file number that the control card is in.
insf	integer	Flag to invoke output of *SET_NODE data
ip1dblt	integer	output of 1D seatbelt created for 2D seatbelt to sbtout
ipcurv	integer	output curve data flag
ipnint	integer	Flag to print initial timesteps at cycle #1
iprtf	integer	Print flag for RBDOUT and MATSUM files
isfcnt	integer	Continuity level in applying interface linking data
isolsf	integer	Flag to invoke output of *SET_SOLID data
issf	integer	Flag to invoke output of *SET_SHELL data
kineng	integer	Flag to output kinetic energy density as a nodal field
minfo	integer	Output penetration information
mlkbag	integer	Flag to invoke output of accumulated airbag mass leakage in ABSTAT
msgflg	integer	Option for printing detail message to d3msg
msgmax	integer	max num messags
neecho	integer	Print suppression during input: echo file
newleg	integer	New legends
npopt	integer	Print suppression during input: printer file
nrefup	integer	Flag to update individual beam 3rd nodes
opifs	real	Output interval for interface file
penout	integer	Flag to output contact penetration for mortar contact
phschng	integer	Message to messag file for phase change on materials 216, 217 and 218
shlsig	integer	Flag to extrapolate stresses for shells with 8 integration points to nodes
solsig	integer	Flag to extrapolate stresses/history variables
spc2bnd	integer	Flag to convert constraints on rigid bodies to equivalent *BOUNDARY_PRESCRIBED_ MOTION_RIGID motion
tet10s8	integer	tet connectivity output
tolev	integer	Timing output levels

*CONTROL_PARALLEL

Properties for *CONTROL_PARALLEL

Name	Type	Description
consty	integer	Consistency (Accuracy) flag
exists	logical	true if control card exists

include	integer	The Include file number that the control card is in.
ncpu	integer	#cpus to use
numrhs	integer	#rh sides written
para	integer	Flag for parallel force assembly

*CONTROL_PORE_AIR

Properties for *CONTROL_PORE_AIR

Name	Type	Description
air_p	real	Pressure of atmospheric air
air_ro	real	Density of atmospheric air
anamsg	integer	Flag to turn off printing of pore air analysis status message
eterm	real	Event termination time
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.

*CONTROL_PORE_FLUID

Properties for *CONTROL_PORE_FLUID

Name	Type	Description
atype	integer	Analysis type
conmax	real	damping factor
conv	real	conduction factor
datum	real	Z elevation of datum
eterm	real	event time termination
etflag	integer	Flag for interpretation of time
exists	logical	true if control card exists
fmax	real	max seepage factor
fmin	real	min seepage factor
ftied	real	Analysis type
grav	real	Gravitational acceleration for Ro.g.h
include	integer	The Include file number that the control card is in.
output	integer	Output flag for stresses
pf_bulk	real	Default bulk modulus of pore fluid
pf_rho	real	Default pore water density
targ	real	target for change of excess pressure
therm	real	thermal vol expansion coeff
tmf	integer	Time magnification factor on seepage. Loadcurve if negative
wtable	real	Default elevation of water table

*CONTROL_PWP_AUTO_TMF

Properties for *CONTROL_PWP_AUTO_TMF

Name	Type	Description
dpwmax	real	Max rate of change of pwp water head (m/s)
exists	logical	true if control card exists
fmax	real	Maximum factor on seepage calc
fmin	real	Minimum factor on seepage calc
include	integer	The Include file number that the control card is in.
sprfac	real	factor for reducing feedback
targ	real	Target max pwp change/thermal timestep

*CONTROL_PZELECTRIC

Properties for *CONTROL_PZELECTRIC

Name	Type	Description
abstol	real	Absolute convergence tolerance
epzmsg	integer	Flag to determine if electric flux and electric field at the element center of piezoelectric material is outputed to D3PLOT
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
maxitr	integer	Maximum number of iterations 2
msgitr	integer	Output iteration message level
ndtrfk	real	Reform the dielectric stiffness matrix every NDTRFK time steps
reltol	real	Relative convergence tolerance
solver	integer	Piezoelectric solver type

*CONTROL_REFERENCE_CONFIGURATION

Properties for *CONTROL_REFERENCE_CONFIGURATION

Name	Type	Description
exists	logical	true if control card exists
include	integer	the Include file number that the control card is in.
iter	integer	iter keyword option
iterfile	string	base name of two files for the ITER keyword option
maxiter	integer	max number of iterations
method	integer	iterative method
step	real	step size used in iterations
targetfile	string	file containing all nodes of the target geometry

tol	real	tolerance used to determine convergence

*CONTROL_REMESHING

Properties for *CONTROL_REMESHING

Name	Type	Description
cid	integer	coordinate system id
dtmin	real	timestep size for remesh
efg	integer	efg keyword option
exists	logical	true if control card exists
iaat	integer	interactive adaptivity adjustable tolerance
iat	integer	interactive adaptivity
iat1	real	tolerance of shear distortion indicator for interactive adaptivity
iat2	real	tolerance of unbalanced nodal distribution indicator for interactive adaptivity
iat3	real	tolerance of volumetric change indicator for interactive adaptivity
icurv	integer	number of elements along radius
ier	integer	remeshing with element erosion
include	integer	The Include file number that the control card is in.
ivt	integer	internal variable transfer in adaptive EFG
mfrac	real	mass ratio gain required for remesh
mm	integer	monotonic mesh resizing
rmax	real	Maximum edge length
rmin	real	Minimum edge length. Loadcurve if negative.
segang	real	angular mesh size in 3-D axisymmetric remeshing
vfloss	real	necessary VF loss for remesh

*CONTROL_RIGID

Properties for *CONTROL_RIGID

Name	Type	Description
exists	logical	true if control card exists
gjadstf	real	Joint rotational stiffness
gjadvsc	real	Joint rotational damping
include	integer	The Include file number that the control card is in.
jntf	integer	Generalized joint stiffness formulation
lmf	integer	Switch explicit/implicit joint formulation
metalf	integer	metalforming option
norbic	integer	Circumvent rigid body inertia check

orthmd	integer	Orthogonalise modes wrt each other
partm	integer	Use global mass matrix for mass distribution
plotel	integer	Automatic generation of *ELEMENT_PLOTEL
rbsms	integer	Flag to apply consistent treatment of rigid bodies in selective mass scaling
rcvlr2d	integer	Recover the lead rigid body of constrained rigid bodies
sparse	integer	Use sparse xply routines for modal & stiffness damping matrices
tjadstf	real	Joint translational stiffness
tjadvsc	real	Joint translational damping

*CONTROL_SHELL

Properties for *CONTROL_SHELL

Name	Type	Description
bwc	integer	Warping stiffness flag for Belytschko-Tsay shells
cntco	integer	include shell ref surface offset
cstyp6	integer	Coord sys for type 6 element
delfr	integer	delete shells where neighbours fail
drcmth	integer	drilling rotation constraint method.
drcpsid	integer	part set for drilling rotation constraint method.
drcpsrm	real	drilling rotation constraint parameter.
esort	integer	Degenerate shell sorting flag (was ITRIST)
excl	integer	.eq.1 if excl above
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
intgrd	integer	Gauss/Lobatto intg rule switch
intperr	integer	Flag for behavior in case of unwanted interp. or extrap. of initial stresses
irnxx	integer	Hughes-Liu shell normal update option
irquad	integer	intg rule
istupd	integer	Shell thickness change option
iswshl	integer	flag for switching between formulations 16 and 30.
itsflg	integer	initial transverse shear stress
keepcs	integer	keep contact segs of failed shells
lamsht	integer	Laminated shell theory update flag
lispsid	integer	Part set ID related to *INITIAL_STRESS_SHELL.
miter	integer	Plane stress plasticity option
nfail1	integer	Flag for distorted 1 intg point shell check
nfail4	integer	Flag for distorted 4 intg point shell check
nlocdt	integer	flag for time step handling for shell elements with offset.
proj	integer	Projection method for warping stiffness

psnfail	integer	part set id for check
psstupd	integer	part set for thichness update, -ve to exclude
rotascl	real	Scale factor for rotary shell mass
sidt4tu	integer	part set for type 4 thickness update where elastic strains are ignored.
stretch	real	Stretch ratio of element diagonals for element deletion
theory	integer	Shell theory to use
tshell	integer	Thermal shell option
wmode	real	W-mode amplitude for element deletion (deg)
wrpang	real	Shell warpage angle (deg)

*CONTROL_SOLID

Properties for *CONTROL_SOLID

Name	Type	Description
coheqc	integer	Cohesive element quality check
esort	integer	Automatic sort of tetra & penta flag
exists	logical	true if control card exists
fmatrix	integer	calculation method for deformation gradient
icoh	integer	global flag for cohesive element deletion
include	integer	The Include file number that the control card is in.
niptets	integer	#intg points for quadratic tets
pm1	integer	10 noded tetrahedral solid node ID 1
pm10	integer	10 noded tetrahedral solid node ID 10
pm2	integer	10 noded tetrahedral solid node ID 2
pm3	integer	10 noded tetrahedral solid node ID 3
pm4	integer	10 noded tetrahedral solid node ID 4
pm5	integer	10 noded tetrahedral solid node ID 5
pm6	integer	10 noded tetrahedral solid node ID 6
pm7	integer	10 noded tetrahedral solid node ID 7
pm8	integer	10 noded tetrahedral solid node ID 8
pm9	integer	10 noded tetrahedral solid node ID 9
psfail	integer	Optional part set id
rinrt	integer	Option to compute rotational inertia for the nodes of solid elements
swlocl	integer	output flag for stresses in solid spotwelds
t10jtol	real	tolerance for jacobian in 4-point 10-noded quadratic tetrahedra
tet13k	integer	global flag for cohesive element deletion
tet13v	integer	Choice of type 13 solid implementation

*CONTROL_SOLUTION

Properties for *CONTROL_SOLUTION

Name	Type	Description
crvp	integer	Bypass time-based evaluation of non-time-dependent curves
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
isnan	integer	Flag to check for a NaN in force and moment arrays
lcacc	integer	Flag to truncate curves: 0 = no truncation; otherwise = truncate
lcint	integer	Number of points in load curve discretization
ncdcf	integer	Cycle number at which to evaluate DEFINE_CURVE_FUNCTION
nlq	integer	Vector length
nocopy	integer	Avoid copying material history variables to temporary buffers for constitutive evaluations
soln	integer	Solution type flag

*CONTROL_SPH

Properties for *CONTROL_SPH

Name	Type	Description
boxid	integer	Box limiting application
cont	integer	Particle approx method
deriv	integer	Time integration type
dt	real	Death time
exists	logical	true if control card exists
form	integer	particle theory
iavis	integer	artificial viscosity formulation
icont	integer	contact option
idim	integer	Space system flag
ierod	integer	erosion option
include	integer	The Include file number that the control card is in.
ini	integer	bucket or global smoothing
ishift	integer	apply shifting algorithm
ishow	integer	display option
istab	integer	stabilisation type
isymp	integer	percentage of sph
ithk	integer	contact thickness option
maxv	real	max velocity
memory	integer	memory alloc
ncbs	integer	Number of cycles between particle sorting
nmneigh	integer	memory alloc

ql	real	quasi-linear coefficient
sphsort	integer	sort and move SPH
start	real	start time

*CONTROL_SPH_INCOMPRESSIBLE

Properties for *CONTROL_SPH_INCOMPRESSIBLE

Name	Type	Description
acmp	real	Artificial compressibility term
exists	logical	true if control card exists
ibndp	integer	Pressure treatment of boundary particles
ihtc	integer	Flag for Heat Transfer Coefficient calculation
imat	integer	Flag for *MAT_SPH_INCOMPRESSIBLE_FLUID formulations
include	integer	The Include file number that the control card is in.
irmv	integer	Flag to remove initially interpenetrated particles
rol	real	Deactivation criteria
tavg	real	Tolerance criteria for average relative density
tmax	real	Tolerance criteria for maximum relative density

*CONTROL_SPOTWELD_BEAM

Properties for *CONTROL_SPOTWELD_BEAM

Name	Type	Description
bmsid	integer	beam set for convert to hex assembly
exists	logical	true if control card exists
id_off	integer	part id offset
include	integer	The Include file number that the control card is in.
lcs	integer	Loadcurve: shear response vs. shell size
lct	integer	Loadcurve: tension response vs. shell size
prtflg	integer	Flag to print data for spotwelds
rpbhx	integer	Replace each beam with a cluster of RPBHX solids
t_ors	integer	Table ID for scaling shear response
t_ort	integer	Table for scaling response

*CONTROL_STAGED_CONSTRUCTION

Properties for *CONTROL_STAGED_CONSTRUCTION

Name	Type	Description

accel	real	gravity
dordel	integer	Dormant part treatment in D3PLOT file
exists	logical	true if control card exists
fact	real	default stiffness/gravity factor
include	integer	The Include file number that the control card is in.
itime	integer	Treatment of "Real time" on *DEFIN_CONSTRUCTION_STAGES
nopdel	integer	Treatment of pressure loads on deleted elements
stge	integer	end stage
stgs	integer	start stage
stref	integer	ref stage
tstart	real	start time

*CONTROL_START

Properties for *CONTROL_START

Name	Type	Description
begtim	real	start time
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.

*CONTROL_STEADY_STATE_ROLLING

Properties for *CONTROL_STEADY_STATE_ROLLING

Name	Type	Description
exists	logical	true if control card exists
imass	integer	Inertia switching flag
include	integer	The Include file number that the control card is in.
ivel	integer	Velocity switching flag
lcdmu	integer	Loadcurve for scaling friction forces
lcdmur	integer	Loadcurve for scaling friction forces during dynamic relaxation
scl_k	integer	Scale factor for friction stiffness

*CONTROL_STRUCTURED

Properties for *CONTROL_STRUCTURED

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.

term	integer	_TERM flag

*CONTROL_TERMINATION

Properties for *CONTROL_TERMINATION

Name	Type	Description
dtmin	real	Scale factor on initial dt size for termination
endcyc	integer	Termination cycle #
endeng	real	%age change in energy for termination
endmas	real	%age change in mass for termination. Loadcurve if negative
endtim	real	Termination time
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
nosol	integer	flag for non-solution run

*CONTROL_THERMAL_EIGENVALUE

Properties for *CONTROL_THERMAL_EIGENVALUE

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
neig	integer	Number of eigen values to compute.

*CONTROL_THERMAL_FORMING

Properties for *CONTROL_THERMAL_FORMING

Name	Type	Description
a	integer	Load curve ID for the a coefficient used in the formula
algo	integer	Contact algorithm type
b	integer	Load curve ID for the b coefficient used in the formula
bc_flg	integer	Thermal boundary condition flag
c	integer	Load curve ID for the c coefficient used in the formula
d	integer	Explicit accuracy parameter
exists	logical	true if control card exists
formula	integer	Formula that defines the contact heat conductance as a function of temperature and pressure
frad	real	Radiation factor between the contact surfaces
ftosa	real	Fraction of sliding friction energy partitioned to the SURFA surface
fwork	real	Fraction of mechanical work converted into heat

h0	real	Heat transfer conductance for closed gaps
include	integer	The Include file number that the control card is in.
ithoff	integer	Flag for offsetting thermal contact surfaces for thick thermal shells
its	real	Initial thermal time step size
k	real	Thermal conductivity of fluid between the contact surfaces
lcfdt	integer	Load curve number for dynamic coefficient of friction as a function of temperature
lcfst	integer	Load curve number for static coefficient of friction as a function of temperature
lch	integer	Load curve ID for h (can be curve ID or function ID)
lmax	real	No thermal contact if gap is greater than this value
lmin	real	Minimum gap
ptype	integer	Thermal problem type
solver	integer	Thermal analysis solver type
thshel	integer	Thermal shell option
tsf	real	Thermal Speedup Factor

*CONTROL_THERMAL_NONLINEAR

Properties for *CONTROL_THERMAL_NONLINEAR

Name	Type	Description
dcp	real	Divergence control parameter
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
lumpbc	integer	lump boundary condition
nlthpr	integer	Thermal nonlinear printout level
phchpn	real	Phase change penalty parameter
refmax	integer	Max #matrix reformations per timestep
thlstl	real	Line search convergence tolerance
tol	real	Convergence tolerance for temperature

*CONTROL_THERMAL_SOLVER

Properties for *CONTROL_THERMAL_SOLVER

Name	Type	Description
abstol	real	Absolute convergence tolerance
atype	integer	Thermal analysis type
cgtol	real	Convergence tolerance for iterative solver
dtvf	real	Time interval between view factor updates
eqheat	integer	Mechanical equivelent of heat (J/Nm etc). Loadcurve if negative

exists	logical	true if control card exists
fwork	real	Fraction of mechanical heat converted into heat
gpt	integer	#gauss points in solids
include	integer	The Include file number that the control card is in.
maxitr	integer	Maximum number of iterations
msglvl	integer	Output message level
mxdmp	integer	Matrix dumping.
ncycl	integer	Thermal matrix reassembly frequency
ninner	integer	Number of inner iterations for GMRES solve
nouter	integer	Number of outer iterations for GMRES solve
omega	real	Relaxation parameter
ptype	integer	Thermal problem type
reltol	real	Relative convergence tolerance
sbc	real	Stefan Boltzman constant (w/m**2/K)
solver	integer	Thermal analysis solver type
tsf	integer	Thermal speedup factor. Loadcurve if negative
varden	integer	Variable thermal density flag

*CONTROL_THERMAL_TIMESTEP

Properties for *CONTROL_THERMAL_TIMESTEP

Name	Type	Description
dtemp	integer	Max delta temp permitted before timestep decrease, of LC of dt vs time if -ve
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
its	real	Initial thermal timestep
lcts	integer	Loadcurve: timestep vs time
tip	real	Thermal time integration parameter
tmax	real	Maximum thermal timestep, or LC of tmax vs time if -ve
tmin	real	Minimum thermal timestep, or LC of tmin vs time if -ve
ts	integer	Thermal timestep control flag
tscp	real	Timestep control parameter

*CONTROL_TIMESTEP

Properties for *CONTROL_TIMESTEP

Name	Type	Description
dt2ms	real	Timestep for mass scaling
dt2msf	real	Scale factor for initial timestep size to determine min permitted time step size

dt2mslc	integer	Loadcurve: DT2MS vs time
dtdynv	integer	Flag to consider the effects of dynamic viscosity on the critical time step
dtinit	real	Initial timestep size
dtusr	integer	User-defined time step for explicit analysis
emscl	real	Fraction of added mass from mass scaling that contributes to gravity loads
erode	integer	Erosion flag for solids & shells @ DTMIN
exists	logical	true if control card exists
igado	integer	Method for calculating time steps for IGA elements
ihdo	integer	Method for calculating solid element time steps
imscl	integer	Selective mass scaling. Part set if negative
include	integer	The Include file number that the control card is in.
isdo	integer	dt calc method for 4 noded shells
lctm	real	Loadcurve: Max timestep vs time
ms1st	integer	Limit mass scaling to 1st timestep flag
rmscl	integer	flag to activate scaling of rotational inertia
tslimt	real	Min timestep for shell modulus change
tssfac	real	Scale factor for computed timestep

*CONTROL_UNITS

Properties for *CONTROL_UNITS

Name	Type	Description
exists	logical	true if control card exists
include	integer	The Include file number that the control card is in.
length	string	m = meter, mm = millimeter, cm = centimeter, in = inch, ft = foot
length_ scale	real	Number of meters in the length unit for the input deck
mass	string	kg = kilogram, g = gram, mg = milligram, lb = pound, slug = pound x sec2/foot, slinch = pound x sec2/inch, mtrc_ton = metric_ton
mass_ scale	real	Number of kilograms in the mass unit for the input deck
temp	string	K = Kelvin, C = Celsius, F = Fahrenheit, R = Rankine
time	string	sec = second, ms = msec/millisec, micro_s = microsec
time_scale	real	Number of seconds in the time unit for the input deck

*CONTROL_VIBRO_ACOUSTIC

Properties for *CONTROL_VIBRO_ACOUSTIC

Name	Type	Description
exists	logical	true if control card exists

include	integer	The Include file number that the control card is in.
ipanelu	integer	Number of strips in U direction
ipanelv	integer	Number of strips in V direction
nmdstr	integer	Number of modes in modal stress/strain output
restrt	integer	Restart option
vaflag	integer	Loading type
vaplot	integer	Flag for PSD broadband plots
vaprld	integer	Flag for including preload
vapsd	integer	Flag for PSD output
varms	integer	Flag for RMS output
vastrs	integer	Flag for including stress analysis

Damping class

The Damping class gives you access to damping cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Damping properties

Name	Type	Description
global	Object	*DAMPING_GLOBAL card
modal	Object	*DAMPING_MODAL card
structural	Object	*DAMPING_STRUCTURAL card

Properties for DAMPING_GLOBAL

Name	Type	Description
exists	logical	true if damping card exists
include	integer	The Include file number that the damping card is in.
lcid	integer	Curve ID specifying system damping constant
srx	real	Scale factor on global x rotational damping moments
sry	real	Scale factor on global y rotational damping moments
srz	real	Scale factor on global z rotational damping moments
stx	real	Scale factor on global x translational damping forces
sty	real	Scale factor on global y translational damping forces
stz	real	Scale factor on global z translational damping forces
valdmp	real	System damping constant

Properties for DAMPING_MODAL

Name	Type	Description
calcdt	real	Calculation interval
exists	logical	true if damping card exists
f_name	real	Filename
include	integer	The Include file number that the damping card is in.
prflg	integer	Printout flag

Properties for DAMPING_STRUCTURAL

Name	Type	Description

exists	logical	true if damping card exists
g	real	Constant structural damping coefficient
include	integer	The Include file number that the damping card is in.
lcid	integer	Curve ID Curve ID to define frequency dependent structural damping coefficients
lctyp	integer	Type of load curve defining structural damping coefficients

Detailed Description

The Damping class allows you to create, modify, edit and manipulate damping cards. Unlike other classes there is no constructor and there are no functions. Instead a Damping object is available as the damping property of a Model object. This object allows you to access the damping cards.
For example,
To activate damping card *DAMPING_GLOBAL in model m and set valdmp to 0.001.

```
m.damping.global.exists = true;
m.damping.global.valdmp = 0.001;
```

To activate damping card *DAMPING_MODAL in model m and set prflg to 1.

```
m.damping.modal.exists = true;
m.damping.modal.prflg = 1;
```

To activate damping card *DAMPING_STRUCTURAL in model m and set g to 0.02.

```
m.damping.structural.exists = true;
m.damping.structural.g = 0.02;
```

See the properties for more details.

DampingFrequencyRange class

The DampingFrequencyRange class gives you access to define damping frequency range cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model *[Model]*, redraw (optional) *[boolean]*)
- BlankFlagged(Model *[Model]*, flag *[Flag]*, redraw (optional) *[boolean]*)
- First(Model *[Model]*)
- FlagAll(Model *[Model]*, flag *[Flag]*)
- ForEach(Model *[Model]*, func *[function]*, extra (optional) *[any]*)
- GetAll(Model *[Model]*, property (optional) *[string]*)
- GetFlagged(Model *[Model]*, flag *[Flag]*, property (optional) *[string]*)
- GetFromID(Model *[Model]*, number *[integer]*)
- Last(Model *[Model]*)
- Pick(prompt *[string]*, limit (optional) *[Model or Flag]*, modal (optional) *[boolean]*, button text (optional) *[string]*)
- Select(flag *[Flag]*, prompt *[string]*, limit (optional) *[Model or Flag]*, modal (optional) *[boolean]*)
- SketchFlagged(Model *[Model]*, flag *[Flag]*, redraw (optional) *[boolean]*)
- Total(Model *[Model]*, exists (optional) *[boolean]*)
- UnblankAll(Model *[Model]*, redraw (optional) *[boolean]*)
- UnblankFlagged(Model *[Model]*, flag *[Flag]*, redraw (optional) *[boolean]*)
- UnflagAll(Model *[Model]*, flag *[Flag]*)
- UnsketchAll(Model *[Model]*, redraw (optional) *[boolean]*)
- UnsketchFlagged(Model *[Model]*, flag *[Flag]*, redraw (optional) *[boolean]*)

Member functions

- AssociateComment(Comment *[Comment]*)
- Blank()
- Blanked()
- ClearFlag(flag *[Flag]*)
- Copy(range (optional) *[boolean]*)
- DetachComment(Comment *[Comment]*)
- Error(message *[string]*, options (optional) *[object]*)
- Error(message *[string]*, details (optional) *[string]*) [deprecated]
- Flagged(flag *[Flag]*)
- GetComments()
- GetParameter(prop *[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag *[Flag]*)
- Sketch(redraw (optional) *[boolean]*)
- Unblank()
- Unsketch(redraw (optional) *[boolean]*)
- ViewParameters()
- Warning(message *[string]*, options (optional) *[object]*)
- Warning(message *[string]*, details (optional) *[string]*) [deprecated]
- Xrefs()
- toString()

DampingFrequencyRange constants

Name	Description
DampingFrequencyRange.DEFORM	DEFORM is *DAMPING_FREQUENCY_RANGE_DEFORM.
DampingFrequencyRange.NO_OPT	NO_OPT is *DAMPING_FREQUENCY_RANGE.

DampingFrequencyRange properties

Name	Type	Description
cdamp	real	Fraction of critical damping
cdampv	real	Fraction of critical damping applied to pressure/volume response of solid elements
exists (read only)	logical	true if damping frequency range exists, false if referred to but not defined.
fhigh	real	Highest frequency in range of interest.
flow	real	Lowest frequency in range of interest
icard2	integer	Flag for reading the additional card
iflg	integer	Method used for internal calculation of damping constants
include	integer	The Include file number that the damping frequency range is in.
ipwp	integer	Flag to determine whether damping is applied to excess pore pressure
model (read only)	integer	The Model number that the damping frequency range is in.
option	constant	The DampingFrequencyRange option. Can be: • DampingFrequencyRange.NO_OPT or 0 • DampingFrequencyRange.DEFORM
pidref	integer	Part ID.
psid	integer	Set Part set ID.

Detailed Description

The DampingFrequencyRange class allows you to create, modify, edit and manipulate damping_frequency_range cards. See the documentation below for more details.

Constructor

new DampingFrequencyRange(Model*[Model]*, option (optional)*[constant]*, cdamp (optional)*[real]*, flow (optional)*[real]*, fhigh (optional)*[real]*, psid (optional)*[integer]*, pidref (optional)*[integer]*, iflg (optional)*[integer]*, icard2 (optional)*[integer]*, cdampv (optional)*[real]*, ipwp (optional)*[integer]*)

Description

Create a new DampingFrequencyRange object for *DAMPING_FREQUENCY_RANGE.

Arguments

- **Model** (Model)

Model that damping frequency range will be created in

- **option (optional)** (constant)

Damping frequency range type. Can be DampingFrequencyRange.NO_OPT or DampingFrequencyRange.DEFORM.

- **cdamp (optional)** (real)

Fraction of critical damping

- **flow (optional)** (real)

Lowest frequency in range of interest

- **fhigh (optional)** (real)

Highest frequency in range of interest.

- **psid (optional)** (integer)

Set Part set ID.

- **pidref (optional)** (integer)

Part ID.

- **iflg (optional)** (integer)

Method used for internal calculation of damping constants

- **icard2 (optional)** (integer)

Flag for reading the additional card

- **cdampv (optional)** (real)

Fraction of critical damping applied to pressure/volume response of solid elements

- **ipwp (optional)** (integer)

Flag to determine whether damping is applied to excess pore pressure

Returns

DampingFrequencyRange object

Return type

DampingFrequencyRange

Example

To create a new damping frequency range (of type DEFORM) in model m with cdamp, flow, fhigh, psid, iflg set to 1.5, 2.5, 3.5, 4, 1 respectively:

```
var fr = new DampingFrequencyRange(m, DampingFrequencyRange.DEFORM, 1.5, 2.5, 3.5, 4, 1);
```

Details of functions

AssociateComment(Comment[*Comment*])

Description

Associates a comment with a damping frequency range.

Arguments

- **Comment** (Comment)

Comment that will be attached to the damping frequency range

Returns

No return value

Example

To associate comment c to the damping frequency range fr:

```
fr.AssociateComment(c);
```

Blank()

Description

Blanks the damping frequency range

Arguments

No arguments

Returns

No return value

Example

To blank damping frequency range fr:

```
fr.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the damping frequency ranges in the model.

Arguments

- **Model** (Model)

Model that all damping frequency ranges will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the damping frequency ranges in model m:

```
DampingFrequencyRange.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged damping frequency ranges in the model.

Arguments

- **Model** (Model)

Model that all the flagged damping frequency ranges will be blanked in

- **flag** (Flag)

Flag set on the damping frequency ranges that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the damping frequency ranges in model m flagged with f:

```
DampingFrequencyRange.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the damping frequency range is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if damping frequency range fr is blanked:

```
if (fr.Blanked() ) do_something...
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the damping frequency range.

Arguments

- **flag** (Flag)

Flag to clear on the damping frequency range

Returns

No return value

Example

To clear flag f for damping frequency range fr:

```
fr.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the damping frequency range. The target include of the copied damping frequency range can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

DampingFrequencyRange object

Return type

DampingFrequencyRange

Example

To copy damping frequency range fr into damping frequency range z:

```
var z = fr.Copy();
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a damping frequency range.

Arguments

- **Comment** (Comment)

Comment that will be detached from the damping frequency range

Returns

No return value

Example

To detach comment c from the damping frequency range fr:

```
fr.DetachComment(c);
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for damping frequency range with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The damping frequency range object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...

autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for damping frequency range fr:

```
fr.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for damping frequency range fr:

```
fr.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for damping frequency range. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for damping frequency range fr:

```
fr.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first damping frequency range in the model.

Arguments

- **Model** (Model)

Model to get first damping frequency range in

Returns

DampingFrequencyRange object (or null if there are no damping frequency ranges in the model).

Return type

DampingFrequencyRange

Example

To get the first damping frequency range in model m:

```
var fr = DampingFrequencyRange.First(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the damping frequency ranges in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all damping frequency ranges will be flagged in

- **flag** (Flag)

Flag to set on the damping frequency ranges

Returns

No return value

Example

To flag all of the damping frequency ranges with flag f in model m:

```
DampingFrequencyRange.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the damping frequency range is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the damping frequency range

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if damping frequency range fr has flag f set on it:

```
if (fr.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each damping frequency range in the model.
Note that ForEach has been designed to make looping over damping frequency ranges as fast as possible and so has some limitations.
Firstly, a single temporary DampingFrequencyRange object is created and on each function call it is updated with the current damping frequency range data. This means that you should not try to store the DampingFrequencyRange object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new damping frequency ranges inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all damping frequency ranges are in

- **func** (function)

Function to call for each damping frequency range

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the damping frequency ranges in model m:

```
DampingFrequencyRange.ForEach(m, test);
function test(fr)
{
// fr is DampingFrequencyRange object
}
```

To call function test for all of the damping frequency ranges in model m with optional object:

```
var data = { x:0, y:0 };
DampingFrequencyRange.ForEach(m, test, data);
function test(fr, extra)
{
// fr is DampingFrequencyRange object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of DampingFrequencyRange objects or properties for all of the damping frequency ranges in a model in PRIMER. If the optional property argument is not given then an array of DampingFrequencyRange objects is returned. If the property argument is given, that property value for each damping frequency range is returned in the array instead of a DampingFrequencyRange object

Arguments

- **Model** (Model)

Model to get damping frequency ranges from

- **property (optional)** (string)

Name for property to get for all damping frequency ranges in the model

Returns

Array of DampingFrequencyRange objects or properties

Return type

Array

Example

To make an array of DampingFrequencyRange objects for all of the damping frequency ranges in model m:

```
var a = DampingFrequencyRange.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each damping frequency range in model m:

```
var a = DampingFrequencyRange.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a damping frequency range.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the damping frequency range fr:

```
var comm_array = fr.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of DampingFrequencyRange objects for all of the flagged damping frequency ranges in a model in PRIMER If the optional property argument is not given then an array of DampingFrequencyRange objects is returned. If the property argument is given, then that property value for each damping frequency range is returned in the array instead of a DampingFrequencyRange object

Arguments

- **Model** (Model)

Model to get damping frequency ranges from

- **flag** (Flag)

Flag set on the damping frequency ranges that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged damping frequency ranges in the model

Returns

Array of DampingFrequencyRange objects or properties

Return type

Array

Example

To make an array of DampingFrequencyRange objects for all of the damping frequency ranges in model m flagged with f:

```
var fr = DampingFrequencyRange.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the damping frequency ranges in model m flagged with f:

```
var a = DampingFrequencyRange.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the DampingFrequencyRange object for a damping frequency range ID.

Arguments

- **Model** (Model)

Model to find the damping frequency range in

- **number** (integer)

number of the damping frequency range you want the DampingFrequencyRange object for

Returns

DampingFrequencyRange object (or null if damping frequency range does not exist).

Return type

DampingFrequencyRange

Example

To get the DampingFrequencyRange object for damping frequency range 100 in model m

```
var fr = DampingFrequencyRange.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a DampingFrequencyRange property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the DampingFrequencyRange.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

damping frequency range property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if DampingFrequencyRange property fr.example is a parameter:

```
Options.property_parameter_names = true;
if (fr.GetParameter(fr.example) ) do_something...
Options.property_parameter_names = false;
```

To check if DampingFrequencyRange property fr.example is a parameter by using the GetParameter method:

```
if (fr.ViewParameters().GetParameter(fr.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this damping frequency range (*DAMPING_FREQUENCY_RANGE). **Note that a carriage return is not added**. See also DampingFrequencyRange.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for damping frequency range m:

```
var key = fr.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the damping frequency range. **Note that a carriage return is not added**. See also DampingFrequencyRange.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for damping frequency range fr:

```
var cards = fr.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last damping frequency range in the model.

Arguments

- **Model** (Model)

Model to get last damping frequency range in

Returns

DampingFrequencyRange object (or null if there are no damping frequency ranges in the model).

Return type

DampingFrequencyRange

Example

To get the last damping frequency range in model m:

```
var fr = DampingFrequencyRange.Last(m);
```

Next()

Description

Returns the next damping frequency range in the model.

Arguments

No arguments

Returns

DampingFrequencyRange object (or null if there are no more damping frequency ranges in the model).

Return type

DampingFrequencyRange

Example

To get the damping frequency range in model m after damping frequency range fr:

```
var fr = fr.Next();
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a damping frequency range.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only damping frequency ranges from that model can be picked. If the argument is a Flag then only damping frequency ranges that are flagged with *limit* can be selected. If omitted, or null, any damping frequency ranges from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

DampingFrequencyRange object (or null if not picked)

Return type

DampingFrequencyRange

Example

To pick a damping frequency range from model m giving the prompt 'Pick damping frequency range from screen':

```
var fr = DampingFrequencyRange.Pick('Pick damping frequency range from screen', m);
```

Previous()

Description

Returns the previous damping frequency range in the model.

Arguments

No arguments

Returns

DampingFrequencyRange object (or null if there are no more damping frequency ranges in the model).

Return type

DampingFrequencyRange

Example

To get the damping frequency range in model m before damping frequency range fr:

```
var fr = fr.Previous();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select damping frequency ranges using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting damping frequency ranges

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only damping frequency ranges from that model can be selected. If the argument is a Flag then only damping frequency ranges that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any damping frequency ranges can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of damping frequency ranges selected or null if menu cancelled

Return type

Number

Example

To select damping frequency ranges from model m, flagging those selected with flag f, giving the prompt 'Select damping frequency ranges':

```
DampingFrequencyRange.Select(f, 'Select damping frequency ranges', m);
```

To select damping frequency ranges, flagging those selected with flag f but limiting selection to damping frequency ranges flagged with flag l, giving the prompt 'Select damping frequency ranges':

```
DampingFrequencyRange.Select(f, 'Select damping frequency ranges', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the damping frequency range.

Arguments

- **flag** (Flag)

Flag to set on the damping frequency range

Returns

No return value

Example

To set flag f for damping frequency range fr:

```
fr.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the damping frequency range. The damping frequency range will be sketched until you either call DampingFrequencyRange.Unsketch(), DampingFrequencyRange.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the damping frequency range is sketched. If omitted redraw is true. If you want to sketch several damping frequency ranges and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch damping frequency range fr:

```
fr.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged damping frequency ranges in the model. The damping frequency ranges will be sketched until you either call DampingFrequencyRange.Unsketch(), DampingFrequencyRange.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged damping frequency ranges will be sketched in

- **flag** (Flag)

Flag set on the damping frequency ranges that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the damping frequency ranges are sketched. If omitted redraw is true. If you want to sketch flagged damping frequency ranges several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all damping frequency ranges flagged with flag in model m:

```
DampingFrequencyRange.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of damping frequency ranges in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing damping frequency ranges should be counted. If false or omitted referenced but undefined damping frequency ranges will also be included in the total.

Returns

number of damping frequency ranges

Return type

Number

Example

To get the total number of damping frequency ranges in model m:

```
var total = DampingFrequencyRange.Total(m);
```

Unblank()

Description

Unblanks the damping frequency range

Arguments

No arguments

Returns

No return value

Example

To unblank damping frequency range fr:

```
fr.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the damping frequency ranges in the model.

Arguments

- **Model** (Model)

Model that all damping frequency ranges will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the damping frequency ranges in model m:

```
DampingFrequencyRange.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged damping frequency ranges in the model.

Arguments

- **Model** (Model)

Model that the flagged damping frequency ranges will be unblanked in

- **flag** (Flag)

Flag set on the damping frequency ranges that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the damping frequency ranges in model m flagged with f:

```
DampingFrequencyRange.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the damping frequency ranges in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all damping frequency ranges will be unset in

- **flag** (Flag)

Flag to unset on the damping frequency ranges

Returns

No return value

Example

To unset the flag f on all the damping frequency ranges in model m:

```
DampingFrequencyRange.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the damping frequency range.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the damping frequency range is unsketched. If omitted redraw is true. If you want to unsketch several damping frequency ranges and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch damping frequency range fr:

```
fr.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all damping frequency ranges.

Arguments

- **Model** (Model)

Model that all damping frequency ranges will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the damping frequency ranges are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all damping frequency ranges in model m:

```
DampingFrequencyRange.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged damping frequency ranges in the model.

Arguments

- **Model** (Model)

Model that all damping frequency ranges will be unsketched in

- **flag** (Flag)

Flag set on the damping frequency ranges that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the damping frequency ranges are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all damping frequency ranges flagged with flag in model m:

```
DampingFrequencyRange.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

DampingFrequencyRange object.

Return type

DampingFrequencyRange

Example

To check if DampingFrequencyRange property fr.example is a parameter by using the DampingFrequencyRange.GetParameter() method:

```
if (fr.ViewParameters().GetParameter(fr.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for damping frequency range with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The damping frequency range object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for damping frequency range fr:

```
fr.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for damping frequency range fr:

```
fr.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for damping frequency range. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for damping frequency range fr:

```
fr.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this damping frequency range.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for damping frequency range fr:

```
var xrefs = fr.Xrefs();
```

---

toString()

Description

Creates a string containing the damping frequency range data in keyword format. Note that this contains the keyword header and the keyword cards. See also DampingFrequencyRange.Keyword() and DampingFrequencyRange.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for damping frequency range fr in keyword format

```
var s = fr.toString();
```

DampingPartMass class

The DampingPartMass class gives you access to define damping part mass cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- First(Model*[Model]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

DampingPartMass constants

Name	Description

DampingPartMass.PART	PART is *DAMPING_PART_MASS.
DampingPartMass.SET	SET is *DAMPING_PART_MASS_SET.

DampingPartMass properties

Name	Type	Description
exists (read only)	logical	true if damping part mass exists, false if referred to but not defined.
flag	integer	Flag for scale factors
id	integer	Part/part set id
include	integer	The Include file number that the damping part mass is in.
lcid	integer	LC: Damping vs time
model (read only)	integer	The Model number that the damping part mass is in.
sf	real	Scale factor on loadcurve
srx	real	Rotational x scale factor
sry	real	Rotational y scale factor
srz	real	Rotational z scale factor
strct	integer	Flag for damping only the structural mass (only deformable bodies) 0: Damp all mass in the part or part set. 1: Damp only structural mass and exclude mass added with *ELEMENT_MASS(_...) from damping. (refers to the STRUCT field - 'strct' in api as 'struct' is reserved)
stx	real	Translational x scale factor
sty	real	Translational y scale factor
stz	real	Translational z scale factor
type	constant	The DampingPartMass type. Can be: • DampingPartMass.PART or • DampingPartMass.SET

Detailed Description

The DampingPartMass class allows you to create, modify, edit and manipulate damping_part_mass cards. See the documentation below for more details.

Constructor

new DampingPartMass(Model*[Model]*, type*[constant]*, id*[integer]*, lcid*[integer]*, sf (optional)*[real]*, flag (optional)*[integer]*, stx (optional)*[real]*, sty (optional)*[real]*, stz (optional)*[real]*, srx (optional)*[real]*, sry (optional)*[real]*, srz (optional)*[real]*)

Description

Create a new DampingPartMass object for *DAMPING_PART_MASS.

Arguments

- **Model** (Model)

Model that damping part mass will be created in

- **type** (constant)

Damping part mass type. Can be DampingPartMass.PART or DampingPartMass.SET.

- **id** (integer)

Part/part set id

- **lcid** (integer)

LC: Damping vs time

- **sf (optional)** (real)

Scale factor on loadcurve

- **flag (optional)** (integer)

Flag for scale factors

- **stx (optional)** (real)

Translational x scale factor

- **sty (optional)** (real)

Translational y scale factor

- **stz (optional)** (real)

Translational z scale factor

- **srx (optional)** (real)

Rotational x scale factor

- **sry (optional)** (real)

Rotational y scale factor

- **srz (optional)** (real)

Rotational z scale factor

Returns

DampingPartMass object

Return type

DampingPartMass

Example

To create a new damping part mass (of type PART) in model m with id, lcid, sf, flag set to 11, 12, 3.5, 1 respectively:

```
var pm = new DampingPartMass(m, DampingPartMass.PART, 11, 12, 3.5, 1);
```

Details of functions

AssociateComment(Comment[*Comment*])

Description

Associates a comment with a damping part mass.

Arguments

- **Comment** (Comment)

Comment that will be attached to the damping part mass

Returns

No return value

Example

To associate comment c to the damping part mass pm:

```
pm.AssociateComment(c);
```

Blank()

Description

Blanks the damping part mass

Arguments

No arguments

Returns

No return value

Example

To blank damping part mass pm:

```
pm.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the damping part masss in the model.

Arguments

- **Model** (Model)

Model that all damping part masss will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the damping part masss in model m:

```
DampingPartMass.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged damping part masss in the model.

Arguments

- **Model** (Model)

Model that all the flagged damping part masss will be blanked in

- **flag** (Flag)

Flag set on the damping part masss that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the damping part masss in model m flagged with f:

```
DampingPartMass.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the damping part mass is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if damping part mass pm is blanked:

```
if (pm.Blanked() ) do_something...
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the damping part mass.

Arguments

- **flag** (Flag)

Flag to clear on the damping part mass

Returns

No return value

Example

To clear flag f for damping part mass pm:

```
pm.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the damping part mass. The target include of the copied damping part mass can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

DampingPartMass object

Return type

DampingPartMass

Example

To copy damping part mass pm into damping part mass z:

```
var z = pm.Copy();
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a damping part mass.

Arguments

- **Comment** (Comment)

Comment that will be detached from the damping part mass

Returns

No return value

Example

To detach comment c from the damping part mass pm:

```
pm.DetachComment(c);
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for damping part mass with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The damping part mass object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for damping part mass pm:

```
pm.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for damping part mass pm:

```
pm.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for damping part mass. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for damping part mass pm:

```
pm.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first damping part mass in the model.

Arguments

- **Model** (Model)

Model to get first damping part mass in

Returns

DampingPartMass object (or null if there are no damping part masss in the model).

Return type

DampingPartMass

Example

To get the first damping part mass in model m:

```
var pm = DampingPartMass.First(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the damping part masss in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all damping part masss will be flagged in

- **flag** (Flag)

Flag to set on the damping part masss

Returns

No return value

Example

To flag all of the damping part masss with flag f in model m:

```
DampingPartMass.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the damping part mass is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the damping part mass

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if damping part mass pm has flag f set on it:

```
if (pm.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each damping part mass in the model.
Note that ForEach has been designed to make looping over damping part masss as fast as possible and so has some limitations.
Firstly, a single temporary DampingPartMass object is created and on each function call it is updated with the current damping part mass data. This means that you should not try to store the DampingPartMass object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new damping part masss inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all damping part masss are in

- **func** (function)

Function to call for each damping part mass

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the damping part masss in model m:

```
DampingPartMass.ForEach(m, test);
function test(pm)
{
// pm is DampingPartMass object
}
```

To call function test for all of the damping part masss in model m with optional object:

```
var data = { x:0, y:0 };
DampingPartMass.ForEach(m, test, data);
function test(pm, extra)
{
// pm is DampingPartMass object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of DampingPartMass objects or properties for all of the damping part masss in a model in PRIMER. If the optional property argument is not given then an array of DampingPartMass objects is returned. If the property argument is given, that property value for each damping part mass is returned in the array instead of a DampingPartMass object

Arguments

- **Model** (Model)

Model to get damping part masss from

- **property (optional)** (string)

Name for property to get for all damping part masss in the model

Returns

Array of DampingPartMass objects or properties

Return type

Array

Example

To make an array of DampingPartMass objects for all of the damping part masss in model m:

```
var a = DampingPartMass.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each damping part mass in model m:

```
var a = DampingPartMass.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a damping part mass.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the damping part mass pm:

```
var comm_array = pm.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of DampingPartMass objects for all of the flagged damping part masss in a model in PRIMER If the optional property argument is not given then an array of DampingPartMass objects is returned. If the property argument is given, then that property value for each damping part mass is returned in the array instead of a DampingPartMass object

Arguments

- **Model** (Model)

Model to get damping part masss from

- **flag** (Flag)

Flag set on the damping part masss that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged damping part masss in the model

Returns

Array of DampingPartMass objects or properties

Return type

Array

Example

To make an array of DampingPartMass objects for all of the damping part masss in model m flagged with f:

```
var pm = DampingPartMass.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the damping part masss in model m flagged with f:

```
var a = DampingPartMass.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the DampingPartMass object for a damping part mass ID.

Arguments

- **Model** (Model)

Model to find the damping part mass in

- **number** (integer)

number of the damping part mass you want the DampingPartMass object for

Returns

DampingPartMass object (or null if damping part mass does not exist).

Return type

DampingPartMass

Example

To get the DampingPartMass object for damping part mass 100 in model m

```
var pm = DampingPartMass.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a DampingPartMass property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the DampingPartMass.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

damping part mass property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if DampingPartMass property pm.example is a parameter:

```
Options.property_parameter_names = true;
if (pm.GetParameter(pm.example) ) do_something...
Options.property_parameter_names = false;
```

To check if DampingPartMass property pm.example is a parameter by using the GetParameter method:

```
if (pm.ViewParameters().GetParameter(pm.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this damping part mass (*DAMPING_PART_MASS). **Note that a carriage return is not added**. See also DampingPartMass.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for damping part mass m:

```
var key = fr.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the damping part mass. **Note that a carriage return is not added**. See also DampingPartMass.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for damping part mass fr:

```
var cards = fr.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last damping part mass in the model.

Arguments

- **Model** (Model)

Model to get last damping part mass in

Returns

DampingPartMass object (or null if there are no damping part masss in the model).

Return type

DampingPartMass

Example

To get the last damping part mass in model m:

```
var pm = DampingPartMass.Last(m);
```

Next()

Description

Returns the next damping part mass in the model.

Arguments

No arguments

Returns

DampingPartMass object (or null if there are no more damping part masss in the model).

Return type

DampingPartMass

Example

To get the damping part mass in model m after damping part mass pm:

```
var pm = pm.Next();
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a damping part mass.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only damping part masss from that model can be picked. If the argument is a Flag then only damping part masss that are flagged with *limit* can be selected. If omitted, or null, any damping part masss from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

DampingPartMass object (or null if not picked)

Return type

DampingPartMass

Example

To pick a damping part mass from model m giving the prompt 'Pick damping part mass from screen':

```
var pm = DampingPartMass.Pick('Pick damping part mass from screen', m);
```

Previous()

Description

Returns the previous damping part mass in the model.

Arguments

No arguments

Returns

DampingPartMass object (or null if there are no more damping part masss in the model).

Return type

DampingPartMass

Example

To get the damping part mass in model m before damping part mass pm:

```
var pm = pm.Previous();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select damping part masss using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting damping part masss

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only damping part masss from that model can be selected. If the argument is a Flag then only damping part masss that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any damping part masss can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of damping part masss selected or null if menu cancelled

Return type

Number

Example

To select damping part masss from model m, flagging those selected with flag f, giving the prompt 'Select damping part masss':

```
DampingPartMass.Select(f, 'Select damping part masss', m);
```

To select damping part masss, flagging those selected with flag f but limiting selection to damping part masss flagged with flag l, giving the prompt 'Select damping part masss':

```
DampingPartMass.Select(f, 'Select damping part masss', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the damping part mass.

Arguments

- **flag** (Flag)

Flag to set on the damping part mass

Returns

No return value

Example

To set flag f for damping part mass pm:

```
pm.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the damping part mass. The damping part mass will be sketched until you either call DampingPartMass.Unsketch(), DampingPartMass.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the damping part mass is sketched. If omitted redraw is true. If you want to sketch several damping part masss and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch damping part mass pm:

```
pm.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged damping part masss in the model. The damping part masss will be sketched until you either call DampingPartMass.Unsketch(), DampingPartMass.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged damping part masss will be sketched in

- **flag** (Flag)

Flag set on the damping part masss that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the damping part masss are sketched. If omitted redraw is true. If you want to sketch flagged damping part masss several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all damping part masss flagged with flag in model m:

```
DampingPartMass.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of damping part masss in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing damping part masss should be counted. If false or omitted referenced but undefined damping part masss will also be included in the total.

Returns

number of damping part masss

Return type

Number

Example

To get the total number of damping part masss in model m:

```
var total = DampingPartMass.Total(m);
```

Unblank()

Description

Unblanks the damping part mass

Arguments

No arguments

Returns

No return value

Example

To unblank damping part mass pm:

```
pm.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the damping part masss in the model.

Arguments

- **Model** (Model)

Model that all damping part masss will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the damping part masss in model m:

```
DampingPartMass.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged damping part masss in the model.

Arguments

- **Model** (Model)

Model that the flagged damping part masss will be unblanked in

- **flag** (Flag)

Flag set on the damping part masss that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the damping part masss in model m flagged with f:

```
DampingPartMass.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the damping part masss in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all damping part masss will be unset in

- **flag** (Flag)

Flag to unset on the damping part masss

Returns

No return value

Example

To unset the flag f on all the damping part masss in model m:

```
DampingPartMass.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the damping part mass.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the damping part mass is unsketched. If omitted redraw is true. If you want to unsketch several damping part masss and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch damping part mass pm:

```
pm.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all damping part masss.

Arguments

- **Model** (Model)

Model that all damping part masss will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the damping part masss are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all damping part masss in model m:

```
DampingPartMass.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged damping part masss in the model.

Arguments

- **Model** (Model)

Model that all damping part masss will be unsketched in

- **flag** (Flag)

Flag set on the damping part masss that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the damping part masss are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all damping part masss flagged with flag in model m:

```
DampingPartMass.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

DampingPartMass object.

Return type

DampingPartMass

Example

To check if DampingPartMass property pm.example is a parameter by using the DampingPartMass.GetParameter() method:

```
if (pm.ViewParameters().GetParameter(pm.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for damping part mass with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The damping part mass object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for damping part mass pm:

```
pm.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for damping part mass pm:

```
pm.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for damping part mass. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for damping part mass pm:

```
pm.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this damping part mass.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for damping part mass pm:

```
var xrefs = pm.Xrefs();
```

---

toString()

Description

Creates a string containing the damping part mass data in keyword format. Note that this contains the keyword header and the keyword cards. See also DampingPartMass.Keyword() and DampingPartMass.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for damping part mass pm in keyword format

```
var s = fr.toString();
```

DampingPartStiffness class

The DampingPartStiffness class gives you access to define damping part stiffness cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- First(Model*[Model]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

DampingPartStiffness constants

Name	Description

DampingPartStiffness.PART	PART is *DAMPING_PART_STIFFNESS.
DampingPartStiffness.SET	SET is *DAMPING_PART_STIFFNESS_SET.

DampingPartStiffness properties

Name	Type	Description
coef	real	Rayleigh damping coefficient
exists (read only)	logical	true if damping part stiffness exists, false if referred to but not defined.
id	integer	Part/part set id
include	integer	The Include file number that the damping part stiffness is in.
model (read only)	integer	The Model number that the damping part stiffness is in.
type	constant	The DampingPartStiffness type. Can be: • DampingPartStiffness.PART or • DampingPartStiffness.SET

Detailed Description

The DampingPartStiffness class allows you to create, modify, edit and manipulate damping_part_stiffness cards. See the documentation below for more details.

Constructor

new DampingPartStiffness(Model*[Model]*, type*[constant]*, id*[integer]*, coef (optional)*[real]*)

Description

Create a new DampingPartStiffness object for *DAMPING_PART_STIFFNESS.

Arguments

- **Model** (Model)

Model that damping part stiffness will be created in

- **type** (constant)

Damping part stiffness type. Can be DampingPartStiffness.PART or DampingPartStiffness.SET.

- **id** (integer)

Part/part set id

- **coef (optional)** (real)

Rayleigh damping coefficient

Returns

DampingPartStiffness object

Return type

DampingPartStiffness

Example

To create a new damping part stiffness (of type PART) in model m with id, coef set to 11, 2.5 respectively:

```
var ps = new DampingPartStiffness(m, DampingPartStiffness.PART, 11, 2.5);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a damping part stiffness.

Arguments

- **Comment** (Comment)

Comment that will be attached to the damping part stiffness

Returns

No return value

Example

To associate comment c to the damping part stiffness ps:

```
ps.AssociateComment(c);
```

Blank()

Description

Blanks the damping part stiffness

Arguments

No arguments

Returns

No return value

Example

To blank damping part stiffness ps:

```
ps.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the damping part stiffnesses in the model.

Arguments

- **Model** (Model)

Model that all damping part stiffnesses will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the damping part stiffnesses in model m:

```
DampingPartStiffness.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged damping part stiffnesses in the model.

Arguments

- **Model** (Model)

Model that all the flagged damping part stiffnesses will be blanked in

- **flag** (Flag)

Flag set on the damping part stiffnesses that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the damping part stiffnesses in model m flagged with f:

```
DampingPartStiffness.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the damping part stiffness is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if damping part stiffness ps is blanked:

```
if (ps.Blanked() ) do_something...
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the damping part stiffness.

Arguments

- **flag** (Flag)

Flag to clear on the damping part stiffness

Returns

No return value

Example

To clear flag f for damping part stiffness ps:

```
ps.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the damping part stiffness. The target include of the copied damping part stiffness can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

DampingPartStiffness object

Return type

DampingPartStiffness

Example

To copy damping part stiffness ps into damping part stiffness z:

```
var z = ps.Copy();
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a damping part stiffness.

Arguments

- **Comment** (Comment)

Comment that will be detached from the damping part stiffness

Returns

No return value

Example

To detach comment c from the damping part stiffness ps:

```
ps.DetachComment(c);
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for damping part stiffness with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The damping part stiffness object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for damping part stiffness ps:

```
ps.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for damping part stiffness ps:

```
ps.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for damping part stiffness. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for damping part stiffness ps:

```
ps.Error("My custom error");
```

First(Model[*Model*]) [static]

Description

Returns the first damping part stiffness in the model.

Arguments

- **Model** (Model)

Model to get first damping part stiffness in

Returns

DampingPartStiffness object (or null if there are no damping part stiffnesses in the model).

Return type

DampingPartStiffness

Example

To get the first damping part stiffness in model m:

```
var ps = DampingPartStiffness.First(m);
```

FlagAll(Model[*Model*], flag[*Flag*]) [static]

Description

Flags all of the damping part stiffnesses in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all damping part stiffnesses will be flagged in

- **flag** (Flag)

Flag to set on the damping part stiffnesses

Returns

No return value

Example

To flag all of the damping part stiffnesses with flag f in model m:

```
DampingPartStiffness.FlagAll(m, f);
```

Flagged(flag[*Flag*])

Description

Checks if the damping part stiffness is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the damping part stiffness

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if damping part stiffness ps has flag f set on it:

```
if (ps.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each damping part stiffness in the model.
Note that ForEach has been designed to make looping over damping part stiffnesses as fast as possible and so has some limitations.
Firstly, a single temporary DampingPartStiffness object is created and on each function call it is updated with the current damping part stiffness data. This means that you should not try to store the DampingPartStiffness object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new damping part stiffnesses inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all damping part stiffnesses are in

- **func** (function)

Function to call for each damping part stiffness

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the damping part stiffnesses in model m:

```
DampingPartStiffness.ForEach(m, test);
function test(ps)
{
// ps is DampingPartStiffness object
}
```

To call function test for all of the damping part stiffnesses in model m with optional object:

```
var data = { x:0, y:0 };
DampingPartStiffness.ForEach(m, test, data);
function test(ps, extra)
{
// ps is DampingPartStiffness object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of DampingPartStiffness objects or properties for all of the damping part stiffnesses in a model in PRIMER. If the optional property argument is not given then an array of DampingPartStiffness objects is returned. If the property argument is given, that property value for each damping part stiffness is returned in the array instead of a DampingPartStiffness object

Arguments

- **Model** (Model)

Model to get damping part stiffnesses from

- **property (optional)** (string)

Name for property to get for all damping part stiffnesses in the model

Returns

Array of DampingPartStiffness objects or properties

Return type

Array

Example

To make an array of DampingPartStiffness objects for all of the damping part stiffnesses in model m:

```
var a = DampingPartStiffness.GetAll(m);
```

To return an array containing the value of property ’foo’ (for example ’x’ for a node) for each damping part stiffness in model m:

```
var a = DampingPartStiffness.GetAll(m, ’foo’);
```

GetComments()

Description

Extracts the comments associated to a damping part stiffness.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the damping part stiffness ps:

```
var comm_array = ps.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of DampingPartStiffness objects for all of the flagged damping part stiffnesses in a model in PRIMER If the optional property argument is not given then an array of DampingPartStiffness objects is returned. If the property argument is given, then that property value for each damping part stiffness is returned in the array instead of a DampingPartStiffness object

Arguments

- **Model** (Model)

Model to get damping part stiffnesses from

- **flag** (Flag)

Flag set on the damping part stiffnesses that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged damping part stiffnesses in the model

Returns

Array of DampingPartStiffness objects or properties

Return type

Array

Example

To make an array of DampingPartStiffness objects for all of the damping part stiffnesses in model m flagged with f:

```
var ps = DampingPartStiffness.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the damping part stiffnesses in model m flagged with f:

```
var a = DampingPartStiffness.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the DampingPartStiffness object for a damping part stiffness ID.

Arguments

- **Model** (Model)

Model to find the damping part stiffness in

- **number** (integer)

number of the damping part stiffness you want the DampingPartStiffness object for

Returns

DampingPartStiffness object (or null if damping part stiffness does not exist).

Return type

DampingPartStiffness

Example

To get the DampingPartStiffness object for damping part stiffness 100 in model m

```
var ps = DampingPartStiffness.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a DampingPartStiffness property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the DampingPartStiffness.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

damping part stiffness property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if DampingPartStiffness property ps.example is a parameter:

```
Options.property_parameter_names = true;
if (ps.GetParameter(ps.example) ) do_something...
Options.property_parameter_names = false;
```

To check if DampingPartStiffness property ps.example is a parameter by using the GetParameter method:

```
if (ps.ViewParameters().GetParameter(ps.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this damping part stiffness (*DAMPING_PART_STIFFNESS). **Note that a carriage return is not added**. See also DampingPartStiffness.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for damping part stiffness m:

```
var key = fr.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the damping part stiffness. **Note that a carriage return is not added**. See also DampingPartStiffness.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for damping part stiffness fr:

```
var cards = fr.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last damping part stiffness in the model.

Arguments

- **Model** (Model)

Model to get last damping part stiffness in

Returns

DampingPartStiffness object (or null if there are no damping part stiffnesses in the model).

Return type

DampingPartStiffness

Example

To get the last damping part stiffness in model m:

```
var ps = DampingPartStiffness.Last(m);
```

Next()

Description

Returns the next damping part stiffness in the model.

Arguments

No arguments

Returns

DampingPartStiffness object (or null if there are no more damping part stiffnesses in the model).

Return type

DampingPartStiffness

Example

To get the damping part stiffness in model m after damping part stiffness ps:

```
var ps = ps.Next();
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a damping part stiffness.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only damping part stiffnesses from that model can be picked. If the argument is a Flag then only damping part stiffnesses that are flagged with *limit* can be selected. If omitted, or null, any damping part stiffnesses from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

DampingPartStiffness object (or null if not picked)

Return type

DampingPartStiffness

Example

To pick a damping part stiffness from model m giving the prompt 'Pick damping part stiffness from screen':

```
var ps = DampingPartStiffness.Pick('Pick damping part stiffness from screen',
m);
```

Previous()

Description

Returns the previous damping part stiffness in the model.

Arguments

No arguments

Returns

DampingPartStiffness object (or null if there are no more damping part stiffnesses in the model).

Return type

DampingPartStiffness

Example

To get the damping part stiffness in model m before damping part stiffness ps:

```
var ps = ps.Previous();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select damping part stiffnesses using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting damping part stiffnesses

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only damping part stiffnesses from that model can be selected. If the argument is a Flag then only damping part stiffnesses that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any damping part stiffnesses can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of damping part stiffnesses selected or null if menu cancelled

Return type

Number

Example

To select damping part stiffnesses from model m, flagging those selected with flag f, giving the prompt 'Select damping part stiffnesses':

```
DampingPartStiffness.Select(f, 'Select damping part stiffnesses', m);
```

To select damping part stiffnesses, flagging those selected with flag f but limiting selection to damping part stiffnesses flagged with flag l, giving the prompt 'Select damping part stiffnesses':

```
DampingPartStiffness.Select(f, 'Select damping part stiffnesses', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the damping part stiffness.

Arguments

- **flag** (Flag)

Flag to set on the damping part stiffness

Returns

No return value

Example

To set flag f for damping part stiffness ps:

```
ps.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the damping part stiffness. The damping part stiffness will be sketched until you either call DampingPartStiffness.Unsketch(), DampingPartStiffness.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the damping part stiffness is sketched. If omitted redraw is true. If you want to sketch several damping part stiffnesses and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch damping part stiffness ps:

```
ps.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged damping part stiffnesses in the model. The damping part stiffnesses will be sketched until you either call DampingPartStiffness.Unsketch(), DampingPartStiffness.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged damping part stiffnesses will be sketched in

- **flag** (Flag)

Flag set on the damping part stiffnesses that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the damping part stiffnesses are sketched. If omitted redraw is true. If you want to sketch flagged damping part stiffnesses several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all damping part stiffnesses flagged with flag in model m:

```
DampingPartStiffness.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of damping part stiffnesses in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing damping part stiffnesses should be counted. If false or omitted referenced but undefined damping part stiffnesses will also be included in the total.

Returns

number of damping part stiffnesses

Return type

Number

Example

To get the total number of damping part stiffnesses in model m:

```
var total = DampingPartStiffness.Total(m);
```

Unblank()

Description

Unblanks the damping part stiffness

Arguments

No arguments

Returns

No return value

Example

To unblank damping part stiffness ps:

```
ps.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the damping part stiffnesses in the model.

Arguments

- **Model** (Model)

Model that all damping part stiffnesses will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the damping part stiffnesses in model m:

```
DampingPartStiffness.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged damping part stiffnesses in the model.

Arguments

- **Model** (Model)

Model that the flagged damping part stiffnesses will be unblanked in

- **flag** (Flag)

Flag set on the damping part stiffnesses that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the damping part stiffnesses in model m flagged with f:

```
DampingPartStiffness.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the damping part stiffnesses in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all damping part stiffnesses will be unset in

- **flag** (Flag)

Flag to unset on the damping part stiffnesses

Returns

No return value

Example

To unset the flag f on all the damping part stiffnesses in model m:

```
DampingPartStiffness.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the damping part stiffness.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the damping part stiffness is unsketched. If omitted redraw is true. If you want to unsketch several damping part stiffnesses and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch damping part stiffness ps:

```
ps.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all damping part stiffnesses.

Arguments

- **Model** (Model)

Model that all damping part stiffnesses will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the damping part stiffnesses are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all damping part stiffnesses in model m:

```
DampingPartStiffness.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged damping part stiffnesses in the model.

Arguments

- **Model** (Model)

Model that all damping part stiffnesses will be unsketched in

- **flag** (Flag)

Flag set on the damping part stiffnesses that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the damping part stiffnesses are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all damping part stiffnesses flagged with flag in model m:

```
DampingPartStiffness.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

DampingPartStiffness object.

Return type

DampingPartStiffness

Example

To check if DampingPartStiffness property ps.example is a parameter by using the DampingPartStiffness.GetParameter() method:

```
if (ps.ViewParameters().GetParameter(ps.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for damping part stiffness with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description

autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The damping part stiffness object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for damping part stiffness ps:

```
ps.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for damping part stiffness ps:

```
ps.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for damping part stiffness. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for damping part stiffness ps:

```
ps.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this damping part stiffness.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for damping part stiffness ps:

```
var xrefs = ps.Xrefs();
```

---

toString()

Description

Creates a string containing the damping part stiffness data in keyword format. Note that this contains the keyword header and the keyword cards. See also DampingPartStiffness.Keyword() and DampingPartStiffness.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for damping part stiffness ps in keyword format

```
var s = fr.toString();
```

DampingPartStructural class

The DampingPartStructural class gives you access to define damping part structural cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- First(Model*[Model]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

DampingPartStructural constants

Name	Description

DampingPartStructural.PART	PART is *DAMPING_PART_STRUCTURAL.
DampingPartStructural.SET	SET is *DAMPING_PART_STRUCTURAL_SET.

DampingPartStructural properties

Name	Type	Description
coef	real	Structural damping coefficient
exists (read only)	logical	true if damping part structural exists, false if referred to but not defined.
id	integer	Part/part set id
include	integer	The Include file number that the damping part structural is in.
model (read only)	integer	The Model number that the damping part structural is in.
type	constant	The DampingPartStructural type. Can be: • DampingPartStructural.PART or • DampingPartStructural.SET

Detailed Description

The DampingPartStructural class allows you to create, modify, edit and manipulate damping_part_structural cards. See the documentation below for more details.

Constructor

new DampingPartStructural(Model*[Model]*, type*[constant]*, id*[integer]*, coef (optional)*[real]*)

Description

Create a new DampingPartStructural object for *DAMPING_PART_STRUCTURAL.

Arguments

- **Model** (Model)

Model that damping part structural will be created in

- **type** (constant)

Damping part structural type. Can be DampingPartStructural.PART or DampingPartStructural.SET.

- **id** (integer)

Part/part set id

- **coef (optional)** (real)

Structural damping coefficient

Returns

DampingPartStructural object

Return type

DampingPartStructural

Example

To create a new damping part structural (of type PART) in model m with id, coef set to 11, 2.5 respectively:

```
var ps = new DampingPartStructural(m, DampingPartStructural.PART, 11, 2.5);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a damping part structural.

Arguments

- **Comment** (Comment)

Comment that will be attached to the damping part structural

Returns

No return value

Example

To associate comment c to the damping part structural ps:

```
ps.AssociateComment(c);
```

Blank()

Description

Blanks the damping part structural

Arguments

No arguments

Returns

No return value

Example

To blank damping part structural ps:

```
ps.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the damping part structurals in the model.

Arguments

- **Model** (Model)

Model that all damping part structurals will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the damping part structurals in model m:

```
DampingPartStructural.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged damping part structurals in the model.

Arguments

- **Model** (Model)

Model that all the flagged damping part structurals will be blanked in

- **flag** (Flag)

Flag set on the damping part structurals that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the damping part structurals in model m flagged with f:

```
DampingPartStructural.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the damping part structural is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if damping part structural ps is blanked:

```
if (ps.Blanked() ) do_something...
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the damping part structural.

Arguments

- **flag** (Flag)

Flag to clear on the damping part structural

Returns

No return value

Example

To clear flag f for damping part structural ps:

```
ps.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the damping part structural. The target include of the copied damping part structural can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

DampingPartStructural object

Return type

DampingPartStructural

Example

To copy damping part structural ps into damping part structural z:

```
var z = ps.Copy();
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a damping part structural.

Arguments

- **Comment** (Comment)

Comment that will be detached from the damping part structural

Returns

No return value

Example

To detach comment c from the damping part structural ps:

```
ps.DetachComment(c);
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for damping part structural with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The damping part structural object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for damping part structural ps:

```
ps.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for damping part structural ps:

```
ps.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for damping part structural. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for damping part structural ps:

```
ps.Error("My custom error");
```

First(Model[*Model*]) [static]

Description

Returns the first damping part structural in the model.

Arguments

- **Model** (Model)

Model to get first damping part structural in

Returns

DampingPartStructural object (or null if there are no damping part structurals in the model).

Return type

DampingPartStructural

Example

To get the first damping part structural in model m:

```
var ps = DampingPartStructural.First(m);
```

FlagAll(Model[*Model*], flag[*Flag*]) [static]

Description

Flags all of the damping part structurals in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all damping part structurals will be flagged in

- **flag** (Flag)

Flag to set on the damping part structurals

Returns

No return value

Example

To flag all of the damping part structurals with flag f in model m:

```
DampingPartStructural.FlagAll(m, f);
```

Flagged(flag[*Flag*])

Description

Checks if the damping part structural is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the damping part structural

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if damping part structural ps has flag f set on it:

```
if (ps.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each damping part structural in the model.
Note that ForEach has been designed to make looping over damping part structurals as fast as possible and so has some limitations.
Firstly, a single temporary DampingPartStructural object is created and on each function call it is updated with the current damping part structural data. This means that you should not try to store the DampingPartStructural object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new damping part structurals inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all damping part structurals are in

- **func** (function)

Function to call for each damping part structural

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the damping part structurals in model m:

```
DampingPartStructural.ForEach(m, test);
function test(ps)
{
// ps is DampingPartStructural object
}
```

To call function test for all of the damping part structurals in model m with optional object:

```
var data = { x:0, y:0 };
DampingPartStructural.ForEach(m, test, data);
function test(ps, extra)
{
// ps is DampingPartStructural object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of DampingPartStructural objects or properties for all of the damping part structurals in a model in PRIMER. If the optional property argument is not given then an array of DampingPartStructural objects is returned. If the property argument is given, that property value for each damping part structural is returned in the array instead of a DampingPartStructural object

Arguments

- **Model** (Model)

Model to get damping part structurals from

- **property (optional)** (string)

Name for property to get for all damping part structurals in the model

Returns

Array of DampingPartStructural objects or properties

Return type

Array

Example

To make an array of DampingPartStructural objects for all of the damping part structurals in model m:

```
var a = DampingPartStructural.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each damping part structural in model m:

```
var a = DampingPartStructural.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a damping part structural.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the damping part structural ps:

```
var comm_array = ps.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of DampingPartStructural objects for all of the flagged damping part structurals in a model in PRIMER If the optional property argument is not given then an array of DampingPartStructural objects is returned. If the property argument is given, then that property value for each damping part structural is returned in the array instead of a DampingPartStructural object

Arguments

- **Model** (Model)

Model to get damping part structurals from

- **flag** (Flag)

Flag set on the damping part structurals that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged damping part structurals in the model

Returns

Array of DampingPartStructural objects or properties

Return type

Array

Example

To make an array of DampingPartStructural objects for all of the damping part structurals in model m flagged with f:

```
var ps = DampingPartStructural.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the damping part structurals in model m flagged with f:

```
var a = DampingPartStructural.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the DampingPartStructural object for a damping part structural ID.

Arguments

- **Model** (Model)

Model to find the damping part structural in

- **number** (integer)

number of the damping part structural you want the DampingPartStructural object for

Returns

DampingPartStructural object (or null if damping part structural does not exist).

Return type

DampingPartStructural

Example

To get the DampingPartStructural object for damping part structural 100 in model m

```
var ps = DampingPartStructural.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a DampingPartStructural property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the DampingPartStructural.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

damping part structural property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if DampingPartStructural property ps.example is a parameter:

```
Options.property_parameter_names = true;
if (ps.GetParameter(ps.example) ) do_something...
Options.property_parameter_names = false;
```

To check if DampingPartStructural property ps.example is a parameter by using the GetParameter method:

```
if (ps.ViewParameters().GetParameter(ps.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this damping part structural (*DAMPING_PART_STRUCTURAL). **Note that a carriage return is not added**. See also DampingPartStructural.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for damping part structural m:

```
var key = fr.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the damping part structural. **Note that a carriage return is not added**. See also DampingPartStructural.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for damping part structural fr:

```
var cards = fr.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last damping part structural in the model.

Arguments

- **Model** (Model)

Model to get last damping part structural in

Returns

DampingPartStructural object (or null if there are no damping part structurals in the model).

Return type

DampingPartStructural

Example

To get the last damping part structural in model m:

```
var ps = DampingPartStructural.Last(m);
```

Next()

Description

Returns the next damping part structural in the model.

Arguments

No arguments

Returns

DampingPartStructural object (or null if there are no more damping part structurals in the model).

Return type

DampingPartStructural

Example

To get the damping part structural in model m after damping part structural ps:

```
var ps = ps.Next();
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a damping part structural.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only damping part structurals from that model can be picked. If the argument is a Flag then only damping part structurals that are flagged with *limit* can be selected. If omitted, or null, any damping part structurals from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

DampingPartStructural object (or null if not picked)

Return type

DampingPartStructural

Example

To pick a damping part structural from model m giving the prompt 'Pick damping part structural from screen':

```
var ps = DampingPartStructural.Pick('Pick damping part structural from screen', m);
```

Previous()

Description

Returns the previous damping part structural in the model.

Arguments

No arguments

Returns

DampingPartStructural object (or null if there are no more damping part structurals in the model).

Return type

DampingPartStructural

Example

To get the damping part structural in model m before damping part structural ps:

```
var ps = ps.Previous();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select damping part structurals using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting damping part structurals

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only damping part structurals from that model can be selected. If the argument is a Flag then only damping part structurals that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any damping part structurals can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of damping part structurals selected or null if menu cancelled

Return type

Number

Example

To select damping part structurals from model m, flagging those selected with flag f, giving the prompt 'Select damping part structurals':

```
DampingPartStructural.Select(f, 'Select damping part structurals', m);
```

To select damping part structurals, flagging those selected with flag f but limiting selection to damping part structurals flagged with flag l, giving the prompt 'Select damping part structurals':

```
DampingPartStructural.Select(f, 'Select damping part structurals', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the damping part structural.

Arguments

- **flag** (Flag)

Flag to set on the damping part structural

Returns

No return value

Example

To set flag f for damping part structural ps:

```
ps.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the damping part structural. The damping part structural will be sketched until you either call DampingPartStructural.Unsketch(), DampingPartStructural.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the damping part structural is sketched. If omitted redraw is true. If you want to sketch several damping part structurals and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch damping part structural ps:

```
ps.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged damping part structurals in the model. The damping part structurals will be sketched until you either call DampingPartStructural.Unsketch(), DampingPartStructural.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged damping part structurals will be sketched in

- **flag** (Flag)

Flag set on the damping part structurals that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the damping part structurals are sketched. If omitted redraw is true. If you want to sketch flagged damping part structurals several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all damping part structurals flagged with flag in model m:

```
DampingPartStructural.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of damping part structurals in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing damping part structurals should be counted. If false or omitted referenced but undefined damping part structurals will also be included in the total.

Returns

number of damping part structurals

Return type

Number

Example

To get the total number of damping part structurals in model m:

```
var total = DampingPartStructural.Total(m);
```

Unblank()

Description

Unblanks the damping part structural

Arguments

No arguments

Returns

No return value

Example

To unblank damping part structural ps:

```
ps.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the damping part structurals in the model.

Arguments

- **Model** (Model)

Model that all damping part structurals will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the damping part structurals in model m:

```
DampingPartStructural.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged damping part structurals in the model.

Arguments

- **Model** (Model)

Model that the flagged damping part structurals will be unblanked in

- **flag** (Flag)

Flag set on the damping part structurals that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the damping part structurals in model m flagged with f:

```
DampingPartStructural.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the damping part structurals in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all damping part structurals will be unset in

- **flag** (Flag)

Flag to unset on the damping part structurals

Returns

No return value

Example

To unset the flag f on all the damping part structurals in model m:

```
DampingPartStructural.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the damping part structural.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the damping part structural is unsketched. If omitted redraw is true. If you want to unsketch several damping part structurals and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch damping part structural ps:

```
ps.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all damping part structurals.

Arguments

- **Model** (Model)

Model that all damping part structurals will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the damping part structurals are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all damping part structurals in model m:

```
DampingPartStructural.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged damping part structurals in the model.

Arguments

- **Model** (Model)

Model that all damping part structurals will be unsketched in

- **flag** (Flag)

Flag set on the damping part structurals that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the damping part structurals are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all damping part structurals flagged with flag in model m:

```
DampingPartStructural.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

DampingPartStructural object.

Return type

DampingPartStructural

Example

To check if DampingPartStructural property ps.example is a parameter by using the DampingPartStructural.GetParameter() method:

```
if (ps.ViewParameters().GetParameter(ps.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for damping part structural with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description

autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The damping part structural object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for damping part structural ps:

```
ps.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for damping part structural ps:

```
ps.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for damping part structural. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for damping part structural ps:

```
ps.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this damping part structural.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for damping part structural ps:

```
var xrefs = ps.Xrefs();
```

---

toString()

Description

Creates a string containing the damping part structural data in keyword format. Note that this contains the keyword header and the keyword cards. See also DampingPartStructural.Keyword() and DampingPartStructural.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for damping part structural ps in keyword format

```
var s = fr.toString();
```

DampingRelative class

The DampingRelative class gives you access to define damping relative cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- First(Model*[Model]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

DampingRelative properties

Name	Type	Description

cdamp	real	Fraction of critical damping
dv2	real	Constant for velocity squared term
exists (read only)	logical	true if damping relative exists, false if referred to but not defined.
freq	real	Target frequency
include	integer	The Include file number that the damping relative is in.
lcid	integer	Fraction of cricitcal damping vs time
model (read only)	integer	The Model number that the damping relative is in.
pidrb	integer	Part Rigid body ID
psid	integer	Set Part set ID.

Detailed Description

The DampingRelative class allows you to create, modify, edit and manipulate damping_relative cards. See the documentation below for more details.

Constructor

new DampingRelative(Model*[Model]*, pidrb*[integer]*, psid*[integer]*, cdamp (optional)*[real]*, freq (optional)*[real]*, dv2 (optional)*[real]*, lcid (optional)*[integer]*)

Description

Create a new DampingRelative object for *DAMPING_RELATIVE.

Arguments

- **Model** (Model)

Model that damping relative will be created in

- **pidrb** (integer)

Part Rigid body ID

- **psid** (integer)

Set Part set ID.

- **cdamp (optional)** (real)

Fraction of critical damping

- **freq (optional)** (real)

Target frequency

- **dv2 (optional)** (real)

Constant for velocity squared term

- **lcid (optional)** (integer)

Fraction of cricitcal damping vs time

Returns

DampingRelative object

Return type

DampingRelative

Example

To create a new damping relative (of no type) in model m with pidrb, psid, cdamp, freq, dv2, lcid set to 10, 20, 3.5, 4.5, 5.5, 60 respectively:

```
var r = new DampingRelative(m, 10, 20, 3.5, 4.5, 5.5, 60);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a damping relative.

Arguments

- **Comment** (Comment)

Comment that will be attached to the damping relative

Returns

No return value

Example

To associate comment c to the damping relative r:

```
r.AssociateComment(c);
```

Blank()

Description

Blanks the damping relative

Arguments

No arguments

Returns

No return value

Example

To blank damping relative r:

```
r.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the damping relatives in the model.

Arguments

- **Model** (Model)

Model that all damping relatives will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the damping relatives in model m:

```
DampingRelative.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged damping relatives in the model.

Arguments

- **Model** (Model)

Model that all the flagged damping relatives will be blanked in

- **flag** (Flag)

Flag set on the damping relatives that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the damping relatives in model m flagged with f:

```
DampingRelative.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the damping relative is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if damping relative r is blanked:

```
if (r.Blanked() ) do_something...
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the damping relative.

Arguments

- **flag** (Flag)

Flag to clear on the damping relative

Returns

No return value

Example

To clear flag f for damping relative r:

```
r.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the damping relative. The target include of the copied damping relative can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

DampingRelative object

Return type

DampingRelative

Example

To copy damping relative r into damping relative z:

```
var z = r.Copy();
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a damping relative.

Arguments

- **Comment** (Comment)

Comment that will be detached from the damping relative

Returns

No return value

Example

To detach comment c from the damping relative r:

```
r.DetachComment(c);
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for damping relative with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The damping relative object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for damping relative r:

```
r.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for damping relative r:

```
r.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for damping relative. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for damping relative r:

```
r.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first damping relative in the model.

Arguments

- **Model** (Model)

Model to get first damping relative in

Returns

DampingRelative object (or null if there are no damping relatives in the model).

Return type

DampingRelative

Example

To get the first damping relative in model m:

```
var r = DampingRelative.First(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the damping relatives in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all damping relatives will be flagged in

- **flag** (Flag)

Flag to set on the damping relatives

Returns

No return value

Example

To flag all of the damping relatives with flag f in model m:

```
DampingRelative.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the damping relative is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the damping relative

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if damping relative r has flag f set on it:

```
if (r.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each damping relative in the model.
Note that ForEach has been designed to make looping over damping relatives as fast as possible and so has some limitations.
Firstly, a single temporary DampingRelative object is created and on each function call it is updated with the current damping relative data. This means that you should not try to store the DampingRelative object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new damping relatives inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all damping relatives are in

- **func** (function)

Function to call for each damping relative

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the damping relatives in model m:

```
DampingRelative.ForEach(m, test);
function test(r)
{
// r is DampingRelative object
}
```

To call function test for all of the damping relatives in model m with optional object:

```
var data = { x:0, y:0 };
DampingRelative.ForEach(m, test, data);
function test(r, extra)
{
// r is DampingRelative object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of DampingRelative objects or properties for all of the damping relatives in a model in PRIMER. If the optional property argument is not given then an array of DampingRelative objects is returned. If the property argument is given, that property value for each damping relative is returned in the array instead of a DampingRelative object

Arguments

- **Model** (Model)

Model to get damping relatives from

- **property (optional)** (string)

Name for property to get for all damping relatives in the model

Returns

Array of DampingRelative objects or properties

Return type

Array

Example

To make an array of DampingRelative objects for all of the damping relatives in model m:

```
var a = DampingRelative.GetAll(m);
```

To return an array containing the value of property ’foo’ (for example ’x’ for a node) for each damping relative in model m:

```
var a = DampingRelative.GetAll(m, ’foo’);
```

GetComments()

Description

Extracts the comments associated to a damping relative.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the damping relative r:

```
var comm_array = r.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of DampingRelative objects for all of the flagged damping relatives in a model in PRIMER If the optional property argument is not given then an array of DampingRelative objects is returned. If the property argument is given, then that property value for each damping relative is returned in the array instead of a DampingRelative object

Arguments

- **Model** (Model)

Model to get damping relatives from

- **flag** (Flag)

Flag set on the damping relatives that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged damping relatives in the model

Returns

Array of DampingRelative objects or properties

Return type

Array

Example

To make an array of DampingRelative objects for all of the damping relatives in model m flagged with f:

```
var r = DampingRelative.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the damping relatives in model m flagged with f:

```
var a = DampingRelative.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the DampingRelative object for a damping relative ID.

Arguments

- **Model** (Model)

Model to find the damping relative in

- **number** (integer)

number of the damping relative you want the DampingRelative object for

Returns

DampingRelative object (or null if damping relative does not exist).

Return type

DampingRelative

Example

To get the DampingRelative object for damping relative 100 in model m

```
var r = DampingRelative.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a DampingRelative property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the DampingRelative.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

damping relative property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if DampingRelative property r.example is a parameter:

```
Options.property_parameter_names = true;
if (r.GetParameter(r.example) ) do_something...
Options.property_parameter_names = false;
```

To check if DampingRelative property r.example is a parameter by using the GetParameter method:

```
if (r.ViewParameters().GetParameter(r.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this damping relative (*DAMPING_RELATIVE). **Note that a carriage return is not added**. See also DampingRelative.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for damping relative m:

```
var key = r.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the damping relative. **Note that a carriage return is not added**. See also DampingRelative.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for damping relative fr:

```
var cards = r.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last damping relative in the model.

Arguments

- **Model** (Model)

Model to get last damping relative in

Returns

DampingRelative object (or null if there are no damping relatives in the model).

Return type

DampingRelative

Example

To get the last damping relative in model m:

```
var r = DampingRelative.Last(m);
```

Next()

Description

Returns the next damping relative in the model.

Arguments

No arguments

Returns

DampingRelative object (or null if there are no more damping relatives in the model).

Return type

DampingRelative

Example

To get the damping relative in model m after damping relative r:

```
var r = r.Next();
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a damping relative.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only damping relatives from that model can be picked. If the argument is a Flag then only damping relatives that are flagged with *limit* can be selected. If omitted, or null, any damping relatives from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

DampingRelative object (or null if not picked)

Return type

DampingRelative

Example

To pick a damping relative from model m giving the prompt 'Pick damping relative from screen':

```
var r = DampingRelative.Pick('Pick damping relative from screen', m);
```

Previous()

Description

Returns the previous damping relative in the model.

Arguments

No arguments

Returns

DampingRelative object (or null if there are no more damping relatives in the model).

Return type

DampingRelative

Example

To get the damping relative in model m before damping relative r:

```
var r = r.Previous();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select damping relatives using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting damping relatives

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only damping relatives from that model can be selected. If the argument is a Flag then only damping relatives that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any damping relatives can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of damping relatives selected or null if menu cancelled

Return type

Number

Example

To select damping relatives from model m, flagging those selected with flag f, giving the prompt 'Select damping relatives':

```
DampingRelative.Select(f, 'Select damping relatives', m);
```

To select damping relatives, flagging those selected with flag f but limiting selection to damping relatives flagged with flag l, giving the prompt 'Select damping relatives':

```
DampingRelative.Select(f, 'Select damping relatives', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the damping relative.

Arguments

- **flag** (Flag)

Flag to set on the damping relative

Returns

No return value

Example

To set flag f for damping relative r:

```
r.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the damping relative. The damping relative will be sketched until you either call DampingRelative.Unsketch(), DampingRelative.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the damping relative is sketched. If omitted redraw is true. If you want to sketch several damping relatives and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch damping relative r:

```
r.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged damping relatives in the model. The damping relatives will be sketched until you either call DampingRelative.Unsketch(), DampingRelative.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged damping relatives will be sketched in

- **flag** (Flag)

Flag set on the damping relatives that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the damping relatives are sketched. If omitted redraw is true. If you want to sketch flagged damping relatives several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all damping relatives flagged with flag in model m:

```
DampingRelative.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of damping relatives in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing damping relatives should be counted. If false or omitted referenced but undefined damping relatives will also be included in the total.

Returns

number of damping relatives

Return type

Number

Example

To get the total number of damping relatives in model m:

```
var total = DampingRelative.Total(m);
```

Unblank()

Description

Unblanks the damping relative

Arguments

No arguments

Returns

No return value

Example

To unblank damping relative r:

```
r.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the damping relatives in the model.

Arguments

- **Model** (Model)

Model that all damping relatives will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the damping relatives in model m:

```
DampingRelative.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged damping relatives in the model.

Arguments

- **Model** (Model)

Model that the flagged damping relatives will be unblanked in

- **flag** (Flag)

Flag set on the damping relatives that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the damping relatives in model m flagged with f:

```
DampingRelative.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the damping relatives in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all damping relatives will be unset in

- **flag** (Flag)

Flag to unset on the damping relatives

Returns

No return value

Example

To unset the flag f on all the damping relatives in model m:

```
DampingRelative.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the damping relative.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the damping relative is unsketched. If omitted redraw is true. If you want to unsketch several damping relatives and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch damping relative r:

```
r.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all damping relatives.

Arguments

- **Model** (Model)

Model that all damping relatives will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the damping relatives are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all damping relatives in model m:

```
DampingRelative.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged damping relatives in the model.

Arguments

- **Model** (Model)

Model that all damping relatives will be unsketched in

- **flag** (Flag)

Flag set on the damping relatives that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the damping relatives are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all damping relatives flagged with flag in model m:

```
DampingRelative.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

DampingRelative object.

Return type

DampingRelative

Example

To check if DampingRelative property r.example is a parameter by using the DampingRelative.GetParameter() method:

```
if (r.ViewParameters().GetParameter(r.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for damping relative with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description

autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The damping relative object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for damping relative r:

```
r.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for damping relative r:

```
r.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for damping relative. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for damping relative r:

```
r.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this damping relative.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for damping relative r:

```
var xrefs = r.Xrefs();
```

---

toString()

Description

Creates a string containing the damping relative data in keyword format. Note that this contains the keyword header and the keyword cards. See also DampingRelative.Keyword() and DampingRelative.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for damping relative r in keyword format

```
var s = r.toString();
```

Database class

The Database class gives you access to database cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Database properties

Name	Type	Description
abstat	Object	*DATABASE_ABSTAT card
atdout	Object	*DATABASE_ATDOUT card
bearing	Object	*DATABASE_BEARING card
binary	Object	*DATABASE_BINARY card
bndout	Object	*DATABASE_BNDOUT card
cable	Object	*DATABASE_CABLE card
dcfail	Object	*DATABASE_DCFAIL card
defgeo	Object	*DATABASE_DEFGEO card
deforc	Object	*DATABASE_DEFORC card
destat	Object	*DATABASE_DESTAT card
elout	Object	*DATABASE_ELOUT card
envelope	Object	*DATABASE_ENVELOPE card
extent_binary	Object	*DATABASE_EXTENT_BINARY card
extent_binary_comp	Object	*DATABASE_EXTENT_BINARY_COMP card
extent_d3part	Object	*DATABASE_EXTENT_D3PART card
extent_intfor	Object	*DATABASE_EXTENT_INTFOR card
format	Object	*DATABASE_FORMAT card
gceout	Object	*DATABASE_GCEOUT card
glstat	Object	*DATABASE_GLSTAT card
glstat_dr	Object	*DATABASE_GLSTAT_DR card
h3out	Object	*DATABASE_H3OUT card
icvout	Object	*DATABASE_DEBOND card
icvout	Object	*DATABASE_ICVOUT card
jntforc	Object	*DATABASE_JNTFORC card
matsum	Object	*DATABASE_MATSUM card
matsum_dr	Object	*DATABASE_MATSUM_DR card

ncforc	Object	*DATABASE_NCFORC card
nodfor	Object	*DATABASE_NODFOR card
nodout	Object	*DATABASE_NODOUT card
pbstat	Object	*DATABASE_PBSTAT card
pllyout	Object	*DATABASE_PLLYOUT card
prtube	Object	*DATABASE_PRTUBE card
pyro	Object	*DATABASE_PYRO card
rbdout	Object	*DATABASE_RBDOUT card
rcforc	Object	*DATABASE_RCFORC card
rcforc_dr	Object	*DATABASE_RCFORC_DR card
rwforc	Object	*DATABASE_RWFORC card
sbtout	Object	*DATABASE_SBTOUT card
secforc	Object	*DATABASE_SECFORC card
sleout	Object	*DATABASE_SLEOUT card
sleout_dr	Object	*DATABASE_SLEOUT_DR card
snsrout	Object	*DATABASE_SNSROUT card
spcforc	Object	*DATABASE_SPCFORC card
spgcpl	Object	*DATABASE_SPGSPL card
sphmassflow	Object	*DATABASE_SPHMASSFLOW card
sphout	Object	*DATABASE_SPHOUT card
swforc	Object	*DATABASE_SWFORC card
tprint	Object	*DATABASE_TPRINT card
trhist	Object	*DATABASE_TRHIST card

Properties for DATABASE_ABSTAT

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output
exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.
ioopt	integer	Flag for behaviour of load curve
lcur	integer	Curve ID specifying time interval

Properties for DATABASE_ATDOUT

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output
exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.

ioopt	integer	Flag for behaviour of load curve
lcur	integer	Curve ID specifying time interval

Properties for DATABASE_BEARING

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output
exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.
ioopt	integer	Flag for behaviour of load curve
lcur	integer	Curve ID specifying time interval

Properties for DATABASE_BINARY

Name	Type	Description
blstfor	Object	*DATABASE_BINARY_BLSTFOR card
cpmfor	Object	*DATABASE_BINARY_CPMFOR card
d3crack	Object	*DATABASE_BINARY_D3CRACK card
d3drlf	Object	*DATABASE_BINARY_D3DRLF card
d3dump	Object	*DATABASE_BINARY_D3DUMP card
d3mean	Object	*DATABASE_BINARY_D3MEAN card
d3part	Object	*DATABASE_BINARY_D3PART card
d3plot	Object	*DATABASE_BINARY_D3PLOT card
d3prop	Object	*DATABASE_BINARY_D3PROP card
d3thdt	Object	*DATABASE_BINARY_D3THDT card
demfor	Object	*DATABASE_BINARY_DEMFOR card
fsifor	Object	*DATABASE_BINARY_FSIFOR card
fsilnk	Object	*DATABASE_BINARY_FSILNK card
intfor	Object	*DATABASE_BINARY_INTFOR card
isphfor	Object	*DATABASE_BINARY_ISPHFOR card
runrsf	Object	*DATABASE_BINARY_RUNRSF card
xtfile	Object	*DATABASE_BINARY_XTFILE card

Properties for DATABASE_BINARY_BLSTFOR

Name	Type	Description
dt	real	Time interval between outputs
exists	logical	true if database binary card exists
include	integer	The Include file number that the database binary card is in.

Properties for DATABASE_BINARY_CPMFOR

Name	Type	Description
dt	real	Time interval between outputs
exists	logical	true if database binary card exists
include	integer	The Include file number that the database binary card is in.

Properties for DATABASE_BINARY_D3CRACK

Name	Type	Description
dt	real	Time interval between outputs
exists	logical	true if database binary card exists
include	integer	The Include file number that the database binary card is in.

Properties for DATABASE_BINARY_D3DRLF

Name	Type	Description
cycl	integer	Output interval in cycles
exists	logical	true if database binary card exists
include	integer	The Include file number that the database binary card is in.

Properties for DATABASE_BINARY_D3DUMP

Name	Type	Description
cycl	integer	Output interval in cycles
exists	logical	true if database binary card exists
include	integer	The Include file number that the database binary card is in.

Properties for DATABASE_BINARY_D3MEAN

Name	Type	Description
dt	real	Time interval between outputs
exists	logical	true if database binary card exists
iavg	integer	Averaging time interval
include	integer	The Include file number that the database binary card is in.
istats	integer	Level of statistics
tstart	real	Start time

Properties for DATABASE_BINARY_D3PART

Name	Type	Description
beam	integer	Beam option
bsetid	integer	Beam Set ID
dt	real	Time interval between outputs
exists	logical	true if database binary card exists
hsetid	integer	Solid Set ID

include	integer	The Include file number that the database binary card is in.
lcdt	integer	Curve ID giving time interval between dumps
npltc	integer	Number of plot files
psetid	integer	Part Set ID
ssetid	integer	Shell Set ID
tsetid	integer	TShell Set ID

Properties for DATABASE_BINARY_D3PLOT

Name	Type	Description
beam	integer	Beam option
cutoff	real	Frequency cut-off C in Hz
dt	real	Time interval between outputs
exists	logical	true if database binary card exists
include	integer	The Include file number that the database binary card is in.
ioopt	integer	Flag for lcdt behaviour
lcdt	integer	Curve ID giving time interval between dumps
npltc	integer	Number of plot files
pset	integer	Part set ID for filtering
psetid	integer	Part Set ID
rate	real	Time interval T between filter sampling
type	integer	Flag for filtering options
window	real	Width of the window in units of time for storing single, forward filtering

Properties for DATABASE_BINARY_D3PROP

Name	Type	Description
exists	logical	true if database binary card exists
ifile	integer	Output data flag
imatl	integer	Output *EOS, *HOURGLASS, *MAT, *part and *SECTION data
include	integer	The Include file number that the database binary card is in.
iwall	integer	Output *RIGIDWALL data

Properties for DATABASE_BINARY_D3THDT

Name	Type	Description
dt	real	Time interval between outputs
exists	logical	true if database binary card exists
include	integer	The Include file number that the database binary card is in.
lcdt	integer	Curve ID giving time interval between dumps

Properties for DATABASE_BINARY_DEMFOR

Name	Type	Description
dt	real	Time interval between outputs
exists	logical	true if database binary card exists
include	integer	The Include file number that the database binary card is in.

Properties for DATABASE_BINARY_FSIFOR

Name	Type	Description
dt	real	Time interval between outputs
exists	logical	true if database binary card exists
include	integer	The Include file number that the database binary card is in.

Properties for DATABASE_BINARY_FSILNK

Name	Type	Description
dt	real	Time interval between outputs
exists	logical	true if database binary card exists
include	integer	The Include file number that the database binary card is in.

Properties for DATABASE_BINARY_INTFOR

Name	Type	Description
dt	real	Time interval between outputs
exists	logical	true if database binary card exists
fname	string	Filename of the database for the INTFOR data
include	integer	The Include file number that the database binary card is in.
ioopt	integer	governs how the plot state frequency is determined from curve LCDT
lcdt	integer	Curve ID giving time interval between dumps

Properties for DATABASE_BINARY_ISPHFOR

Name	Type	Description
dt	real	Time interval between outputs
exists	logical	true if database binary card exists
include	integer	The Include file number that the database binary card is in.

Properties for DATABASE_BINARY_RUNRSF

Name	Type	Description
cycl	integer	Output interval in cycles
exists	logical	true if database binary card exists
include	integer	The Include file number that the database binary card is in.
nr	integer	Number of running restart files

Properties for DATABASE_BINARY_XTFILE

Name	Type	Description
dt	real	Time interval between outputs
exists	logical	true if database binary card exists
include	integer	The Include file number that the database binary card is in.

Properties for DATABASE_BNDOUT

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output
exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.
ioopt	integer	Flag for behaviour of load curve
lcur	integer	Curve ID specifying time interval

Properties for DATABASE_CABLE

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output
exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.
ioopt	integer	Flag for behaviour of load curve
lcur	integer	Curve ID specifying time interval

Properties for DATABASE_CURVOUT

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output
exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.
ioopt	integer	Flag for behaviour of load curve
lcur	integer	Curve ID specifying time interval

Properties for DATABASE_DCFAIL

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output
exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.
ioopt	integer	Flag for behaviour of load curve

lcur	integer	Curve ID specifying time interval

Properties for DATABASE_DEBOND

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output
exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.
ioopt	integer	Flag for behaviour of load curve
lcur	integer	Curve ID specifying time interval

Properties for DATABASE_DEFGEO

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output
exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.
ioopt	integer	Flag for behaviour of load curve
lcur	integer	Curve ID specifying time interval

Properties for DATABASE_DEFORC

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output
exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.
ioopt	integer	Flag for behaviour of load curve
lcur	integer	Curve ID specifying time interval

Properties for DATABASE_DESTAT

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output
exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.
ioopt	integer	Flag for behaviour of load curve
lcur	integer	Curve ID specifying time interval

Properties for DATABASE_DISBOUT

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output
exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.
ioopt	integer	Flag for behaviour of load curve
lcur	integer	Curve ID specifying time interval

Properties for DATABASE_ELOUT

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output
exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.
ioopt	integer	Flag for behaviour of load curve
lcur	integer	Curve ID specifying time interval
option1	integer	extra history variables for solids
option2	integer	extra history variables for shells
option3	integer	extra history variables for thick shells
option4	integer	extra history variables for beams

Properties for DATABASE_ENVELOPE

Name	Type	Description
bsetid	integer	Output for beam elements. +n is output for elements in beam set n, 0 no beam, -1 all elements
exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.
output	integer	Output format. Can be 0 or 1
ssetid	integer	Output for shell elements. +n is output for elements in shell set n, 0 no shell, -1 all elements
tback	real	Time interval for backup output files during the analysis
tcheck	real	Time interval for checking whether the previous maxima/minima are exceeded

Properties for DATABASE_EXTENT_BINARY

Name	Type	Description
beamip	integer	#beam int points to output
cmpflg	integer	Flag to output composite material stress in local csys
cubsld	integer	Output flag for quadratic solid types
dcomp	integer	Data compression flag
deleres	integer	Output flag for results of deleted elements
dtdt	integer	output of nodal temp

engflg	integer	Flag to in/exclude shell energy & thickness
epsflg	integer	Flag to in/exclude shell strains
exists	logical	true if database card exists
hydro	integer	adds extra history variables
ialemat	integer	output ale materials
ieverp	integer	Every D3PLOT file to separate database flag
include	integer	The Include file number that the database card is in.
intout	string	output of intg pt data
maxint	integer	#integration points for shell output
msscl	integer	output nodal mass scaling data
n3thdt	integer	Output for material energies to D3THDT file
neipb	integer	Output of loop-stresses to D3PLOT
neiph	integer	#extra values for solids
neips	integer	#extra values for shells
nintsld	integer	number of solid integration pts
nodout	string	output of connectivity nodes
pkp_sen	integer	Flag to output peak pressure and surface energy for each contact interface
quadsld	integer	Output flag for cubic solid types
resplt	integer	Output of residual forces
rltflg	integer	Flag to in/exclude shell force/moment resultants
sclp	real	Scaling parameter used in the computation of the peak pressure
shge	integer	Shell hourglass energy output flag
sigflg	integer	Flag to in/exclude shell stress tensors
strflg	integer	Strain tensor output flag
stssz	integer	Output shell element dt flag
therm	integer	Output of thermal data to D3PLOT

Properties for DATABASE_EXTENT_BINARY_COMP

Name	Type	Description
exists	logical	true if database card exists
iacc	string	output of accleration data
iglb	string	output of global data
include	integer	The Include file number that the database card is in.
ised	string	output of strain energy density data
istra	string	output of strain data
istrs	string	output of stress data
ivel	string	output of velocity data
ixyz	string	output of geometry data

Properties for DATABASE_EXTENT_D3PART

Name	Type	Description
engflg	integer	Flag to in/exclude shell energy & thickness
epsflg	integer	Flag to in/exclude shell strains
exists	logical	true if database card exists
ieverp	integer	Every D3PLOT file to separate database flag
include	integer	The Include file number that the database card is in.
maxint	integer	#integration points for shell output
neiph	integer	#extra values for solids
neips	integer	#extra values for shells
nintsld	integer	number of solid integration pts
rltflg	integer	Flag to in/exclude shell force/moment resultants
shge	integer	Shell hourglass energy output flag
sigflg	integer	Flag to in/exclude shell stress tensors
strflg	integer	Strain tensor output flag
stssz	integer	Output shell element dt flag

Properties for DATABASE_EXTENT_INTFOR

Name	Type	Description
exists	logical	true if database card exists
ieverf	integer	Every INTFOR database to separate file flag
include	integer	The Include file number that the database card is in.
neng	integer	Output contact energy density for mortar contact and SOFT = 2 contact
nfail	integer	Display deleted contact segments flag
nforc	integer	Output forces
ngapc	integer	Output contact gaps
nglbv	integer	Output global variables
nhuf	integer	Number of user friction history variables to output from user defined friction routines
npen	integer	Output penetration information for mortar contact
npresu	integer	Output pressures
nshear	integer	Output shear stresses
ntied	integer	Output tied segments for Mortar contact
nvelo	integer	Output nodal velocity
nwear	integer	Output contact wear data mode
nwrk	integer	Output (total) sliding interface energy density for mortar contact
nwusr	integer	Number of user wear history variables

Properties for DATABASE_FORMAT

Name	Type	Description

exists	logical	true if database card exists
ibinary	integer	Word size for binary output files
iform	integer	Output format for D3PLOT and D3THDT files
include	integer	The Include file number that the database card is in.

Properties for DATABASE_GCEOUT

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output
exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.
ioopt	integer	Flag for behaviour of load curve
lcur	integer	Curve ID specifying time interval

Properties for DATABASE_GLSTAT

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output
exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.
ioopt	integer	Flag for behaviour of load curve
lcur	integer	Curve ID specifying time interval
mass_properties	integer	Flag to include mass and inertia properties

Properties for DATABASE_GLSTAT_DR

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output
exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.
ioopt	integer	Flag for behaviour of load curve
lcur	integer	Curve ID specifying time interval
mass_properties	integer	Flag to include mass and inertia properties

Properties for DATABASE_H3OUT

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output
exists	logical	true if database card exists

include	integer	The Include file number that the database card is in.
ioopt	integer	Flag for behaviour of load curve
lcur	integer	Curve ID specifying time interval

Properties for DATABASE_ICVOUT

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output
exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.
ioopt	integer	Flag for behaviour of load curve
lcur	integer	Curve ID specifying time interval

Properties for DATABASE_JNTFORC

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output
exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.
ioopt	integer	Flag for behaviour of load curve
lcur	integer	Curve ID specifying time interval

Properties for DATABASE_MATSUM

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output
exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.
ioopt	integer	Flag for behaviour of load curve
lcur	integer	Curve ID specifying time interval

Properties for DATABASE_MATSUM_DR

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output
exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.
ioopt	integer	Flag for behaviour of load curve
lcur	integer	Curve ID specifying time interval

Properties for DATABASE_NCFORC

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output
exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.
ioopt	integer	Flag for behaviour of load curve
lcur	integer	Curve ID specifying time interval

Properties for DATABASE_NODFOR

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output
exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.
ioopt	integer	Flag for behaviour of load curve
lcur	integer	Curve ID specifying time interval

Properties for DATABASE_NODOUT

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output
exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.
ioopt	integer	Flag for behaviour of load curve
lcur	integer	Curve ID specifying time interval
option1	real	High frequency output interval
option2	integer	Flag for binary file for high frequency output

Properties for DATABASE_PBSTAT

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output
exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.
ioopt	integer	Flag for behaviour of load curve
lcur	integer	Curve ID specifying time interval

Properties for DATABASE_PLLYOUT

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output
exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.
ioopt	integer	Flag for behaviour of load curve
lcur	integer	Curve ID specifying time interval

Properties for DATABASE_PRTUBE

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output
exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.
ioopt	integer	Flag for behaviour of load curve
lcur	integer	Curve ID specifying time interval

Properties for DATABASE_PYRO

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output
exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.
ioopt	integer	Flag for behaviour of load curve
lcur	integer	Curve ID specifying time interval

Properties for DATABASE_RBDOUT

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output
exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.
ioopt	integer	Flag for behaviour of load curve
lcur	integer	Curve ID specifying time interval

Properties for DATABASE_RCFORC

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output

exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.
ioopt	integer	Flag for behaviour of load curve
lcur	integer	Curve ID specifying time interval

Properties for DATABASE_RCFORC_DR

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output
exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.
ioopt	integer	Flag for behaviour of load curve
lcur	integer	Curve ID specifying time interval

Properties for DATABASE_RWFORC

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output
exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.
ioopt	integer	Flag for behaviour of load curve
lcur	integer	Curve ID specifying time interval

Properties for DATABASE_SBTOUT

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output
exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.
ioopt	integer	Flag for behaviour of load curve
lcur	integer	Curve ID specifying time interval

Properties for DATABASE_SECFORC

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output
exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.
ioopt	integer	Flag for behaviour of load curve

lcur	integer	Curve ID specifying time interval

Properties for DATABASE_SLEOUT

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output
exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.
ioopt	integer	Flag for behaviour of load curve
lcur	integer	Curve ID specifying time interval

Properties for DATABASE_SLEOUT_DR

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output
exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.
ioopt	integer	Flag for behaviour of load curve
lcur	integer	Curve ID specifying time interval

Properties for DATABASE_SNSROUT

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output
exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.
ioopt	integer	Flag for behaviour of load curve
lcur	integer	Curve ID specifying time interval

Properties for DATABASE_SPCFORC

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output
exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.
ioopt	integer	Flag for behaviour of load curve
lcur	integer	Curve ID specifying time interval

Properties for DATABASE_SPGCPL

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output
exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.
ioopt	integer	Flag for behaviour of load curve
lcur	integer	Curve ID specifying time interval

Properties for DATABASE_SPHMASSFLOW

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output
exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.
ioopt	integer	Flag for behaviour of load curve
lcur	integer	Curve ID specifying time interval

Properties for DATABASE_SPHOUT

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output
exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.
ioopt	integer	Flag for behaviour of load curve
lcur	integer	Curve ID specifying time interval

Properties for DATABASE_SWFORC

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output
exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.
ioopt	integer	Flag for behaviour of load curve
lcur	integer	Curve ID specifying time interval

Properties for DATABASE_TPRINT

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output

exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.
ioopt	integer	Flag for behaviour of load curve
lcur	integer	Curve ID specifying time interval

Properties for DATABASE_TRHIST

Name	Type	Description
binary	integer	Flag for binary file
dt	real	Time interval between output
exists	logical	true if database card exists
include	integer	The Include file number that the database card is in.
ioopt	integer	Flag for behaviour of load curve
lcur	integer	Curve ID specifying time interval

Detailed Description

The Database class allows you to create, modify, edit and manipulate database cards. Unlike other classes there is no constructor and there are no functions. Instead a Database object is available as the database property of a Model object. This object allows you to access all of the database cards. For example, to activate database card *DATABASE_SWFORC in model m and set dt to 0.001.

```
m.database.swforc.exists = true;
m.database.swforc.dt = 0.001;
```

To activate database card *DATABASE_BINARY_D3PLOT in model m and set dt to 0.001.

```
m.database.binary.d3plot.exists = true;
m.database.binary.d3plot.dt = 0.001;
```

See the properties for more details.

CrossSection class

The CrossSection class gives you access to database cross section cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- CreateAlongFeatureLine(Model*[Model]*, nid1*[integer]*, options*[object]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Autosize(options (optional)*[object]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- ElemCut(Shell label*[integer]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- ExtractColour()
- FlagCut(Flag*[Flag]*)
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- PartCut(Part label*[integer]*, Flag (optional)*[Flag]*)
- Previous()

- Properties()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

CrossSection constants

Name	Description
CrossSection.PLANE	PLANE is *DATABASE_CROSS_SECTION_PLANE.
CrossSection.SET	SET is *DATABASE_CROSS_SECTION_SET.

CrossSection properties

Name	Type	Description
bsid	integer	Beam set number.
colour	Colour	The colour of the cross section
csid	integer	Database cross section number (identical to label).
dsid	integer	Discrete set number.
exists (read only)	logical	true if cross section exists, false if referred to but not defined.
heading	string	Database cross section heading.
hsid	integer	Solid set number.
id	integer	Rigid part or accelerometer or coordinate system number.
idset	logical	true if _ID option is set, false if not
include	integer	The Include file number that the cross section is in.
itype	integer	Flag for local system type.
label	integer	Database cross section number.
lenl	real	Length of L edge.
lenm	real	Length of M edge.
model (read only)	integer	The Model number that the cross section is in.
nsid	integer	Node set number.
option	constant	The Database CrossSection option. Can be: • CrossSection.PLANE or • CrossSection.SET
psid	integer	Part set number.
radius	real	Radius.
ssid	integer	Shell set number.
tsid	integer	Thick shell set number.
xch	real/integer	Head X coord of N normal vector. If <radius> is negative <xch> is a node ID which, combined with <xct>, defines the normal vector of the cut plane.

xct	real/integer	Tail X coord of N normal vector. If <radius> is negative <xct> is a node ID whose coordinates define the centre of the circular cut plane.
xhev	real	Head X coord of L edge vector.
ych	real	Head Y coord of N normal vector.
yct	real	Tail Y coord of N normal vector.
yhev	real	Head Y coord of L edge vector.
zch	real	Head Z coord of N normal vector.
zct	real	Tail Z coord of N normal vector.
zhev	real	Head Z coord of L edge vector.

Detailed Description

The CrossSection class allows you to create, modify, edit and manipulate database cross section cards. See the documentation below for more details.

Constructor

new CrossSection(Model*[Model]*, option*[constant]*, nsid*[integer]*, hsid*[integer]*, bsid*[integer]*, ssid*[integer]*, tsid*[integer]*, dsid*[integer]*, id (optional)*[integer]*, itype (optional)*[integer]*, csid (optional)*[integer]*, heading (optional)*[string]*) [deprecated]

This function is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Create a new CrossSection object for *DATABASE_CROSS_SECTION_SET.

Arguments

- **Model** (Model)

Model that database cross section will be created in

- **option** (constant)

Database cross section type. Must be CrossSection.SET

- **nsid** (integer)

Node set number.

- **hsid** (integer)

Solid set number.

- **bsid** (integer)

Beam set number.

- **ssid** (integer)

Shell set number.

- **tsid** (integer)

Thick shell set number.

- **dsid** (integer)

Discrete set number.

- **id (optional)** (integer)

Rigid part or accelerometer or coordinate system number.

- **itype (optional)** (integer)

Flag for local system type.

- **csid (optional)** (integer)

Database cross_section number.

- **heading (optional)** (string)

Database cross_section title.

Returns

CrossSection object

Return type

CrossSection

Example

To create a new Database cross section 500 called "test cross_section" of type _SET in model m with nsid, hsid, bsid, ssid, tsid, dsid, id, itype set to 11, 12, 13, 14, 15, 16, 17, 2 respectively:

```
var c = new CrossSection(m, CrossSection.SET, 11, 12, 13, 14, 15, 16, 17, 2,
500, "test cross_section");
```

new CrossSection(Model*[Model]*, option*[constant]*, psid*[integer]*, xct*[integer or real]*, yct*[real]*, zct*[real]*, xch*[integer or real]*, ych*[real]*, zch*[real]*, xhev*[real]*, yhev*[real]*, zhev*[real]*, lenl (optional)*[real]*, lenm (optional)*[real]*, id (optional)*[integer]*, itype (optional)*[integer]*, csid (optional)*[integer]*, heading (optional)*[string]*, autosize (optional)*[boolean]*, autosize_pct (optional)*[real]*) [deprecated]

This function is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Create a new CrossSection object for *DATABASE_CROSS_SECTION_PLANE.

Arguments

- **Model** (Model)

Model that database cross section will be created in

- **option** (constant)

Database cross section type. Must be CrossSection.PLANE

- **psid** (integer)

Part set number.

- **xct** (integer or real)

X coordinate of tail of normal vector. If radius < 0.0 then this is the node id for the tail vector.

- **yct** (real)

Y coordinate of tail of normal vector.

- **zct** (real)

Z coordinate of tail of normal vector.

- **xch** (integer or real)

X coordinate of head of normal vector. If radius < 0.0 then this is the node id for the head vector.

- **ych** (real)

Y coordinate of head of normal vector.

- **zch** (real)

Z coordinate of head of normal vector.

- **xhev** (real)

X coordinate of head of edge vector.

- **yhev** (real)

Y coordinate of head of edge vector.

- **zhev** (real)

Z coordinate of head of edge vector.

- **lenl (optional)** (real)

Length in l direction.

- **lenm (optional)** (real)

Length in m direction.

- **id (optional)** (integer)

Rigid part or accelerometer or coordinate system number.

- **itype (optional)** (integer)

Flag for local system type.

- **csid (optional)** (integer)

Database cross_section number.

- **heading (optional)** (string)

Database cross_section title.

- **autosize (optional)** (boolean)

true if the Database cross_section is to be autosized.

- **autosize_pct (optional)** (real)

Additional post-autosize scaling percentage.

Returns

CrossSection object

Return type

CrossSection

Example

To create a new Database cross section 500 called "test cross_section" of type _PLANE in model m with part set ID 100, normal tail (10, 20, 30), normal head (20, 20, 30), head of edge vector (10, 30, 30) and edge lengths 50 and 100:

```
var c = new CrossSection(m, CrossSection.PLANE, 100, 10, 20, 30, 20, 20, 30, 10,
30, 30, 50, 100, 0, 0, 500, "test cross_section");
```

new CrossSection(Model*[Model]*, option*[constant]*, settings*[object]*)

Description

Create a new CrossSection object.

Arguments

- **Model** (Model)

Model that database cross section will be created in

- **option** (constant)

Database cross section type. Must be CrossSection.SET or CrossSection.PLANE

- **settings** (object)

Options specifying various properties used to create the keyword. If optional values are not specified then their default values will be used.

Object has the following properties:

Name	Type	Description
autosize (optional) (PLANE only)	boolean	true if the Database cross_section is to be autosized.
autosize_pct (optional) (PLANE only)	real	Additional post-autosize scaling percentage. (default: same as value specified by the preference primer*cross_section_auto_size_percent)
bsid (SET only)	integer	Beam set number.
csid (optional)	integer	Database cross_section number.
dsid (SET only)	integer	Discrete set number.
heading (optional)	string	Database cross_section title.
hsid (SET only)	integer	Solid set number.
id (optional)	integer	Rigid part or accelerometer or coordinate system number.
itype (optional)	integer	Flag for local system type.
lenl (optional) (PLANE only)	real	Length in l direction.
lenm (optional) (PLANE only)	real	Length in m direction.
nsid (SET only)	integer	Node set number.
psid (PLANE only)	integer	Part set number.
radius (optional)	real	Database cross_section radius. If it is negative xct and xch should be Node ids
ssid (SET only)	integer	Shell set number.
tsid (SET only)	integer	Thick shell set number.
vis_only (optional) (PLANE only)	real	true if autosizing should ignore non-visible elements, false otherwise. (default: true)
xch (PLANE only)	integer or real	X coordinate of head of normal vector. If radius < 0.0 then this is the node id for the head vector.
xct (PLANE only)	integer or real	X coordinate of tail of normal vector. If radius < 0.0 then this is the node id for the tail vector.
xhev (PLANE only)	real	X coordinate of head of edge vector.
ych (PLANE only)	real	Y coordinate of head of normal vector.
yct (PLANE only)	real	Y coordinate of tail of normal vector.
yhev (PLANE only)	real	Y coordinate of head of edge vector.
zch (PLANE only)	real	Z coordinate of head of normal vector.
zct (PLANE only)	real	Z coordinate of tail of normal vector.
zhev (PLANE only)	real	Z coordinate of head of edge vector.

Returns

CrossSection object

Return type

CrossSection

Example

To create a new Database cross section 500 called "test cross_section" of type _SET in model m with nsid, hsid, bsid, ssid, tsid, dsid, id, itype set to 11, 12, 13, 14, 15, 16, 17, 2 respectively:

```
var settings = { nsid: 11, hsid: 12, bsid: 13, ssid: 14, tsid: 15, dsid: 16, id:
17, itype: 2, heading: "test_cross_section" };
var c = new CrossSection(m, CrossSection.SET, settings);
```

To create a new Database cross section 500 called "test cross_section" of type _PLANE in model m with part set ID 100, normal tail (10, 20, 30), normal head (20, 20, 30), head of edge vector (10, 30, 30) and edge lengths 50 and 100:

```
var settings = { psid: 100, id: 500, heading: "test cross_section",
                 xct: 10, yct: 20, zct: 30,
                 xch: 20, ych: 20, zch: 30,
                 xhev: 10, yhev: 30, zhev: 30,
                 lenl: 50, lenm: 100 };
var c = new CrossSection(m, CrossSection.PLANE, settings);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a cross section.

Arguments

- **Comment** (Comment)

Comment that will be attached to the cross section

Returns

No return value

Example

To associate comment c to the cross section c:

```
c.AssociateComment(c);
```

Autosize(options (optional)*[object]*)

Description

Autosizes a _PLANE cross section such that it cuts through all elements in model/psid along that plane.

Arguments

- **options (optional)** (object)

Object containing additional options

Object has the following properties:

Name	Type	Description
percentage (optional)	real	Additional percentage that the autosized cross section will be scaled by. (default: same as value specified by the preference primer*cross_section_auto_size_percent)
vis_only (optional)	boolean	If true, autosizing will ignore non-visible cut elements, false otherwise. (default: true)

Returns

No return value

Example

To autosize a cross section:

```
var xsec = CrossSection.GetFromID(model, id);
if(xsec.option == CrossSection.PLANE) xsec.Autosize();
```

To autosize a cross section, scaled by an additional 5 percent, and ignoring element visiblity:

```
var xsec = CrossSection.GetFromID(model, id);
var options = { percentage: 5.0, vis_only: false };
if(xsec.option == CrossSection.PLANE) xsec.Autosize(options);
```

Blank()

Description

Blanks the cross section

Arguments

No arguments

Returns

No return value

Example

To blank cross section c:

```
c.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the cross sections in the model.

Arguments

- **Model** (Model)

Model that all cross sections will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the cross sections in model m:

```
CrossSection.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged cross sections in the model.

Arguments

- **Model** (Model)

Model that all the flagged cross sections will be blanked in

- **flag** (Flag)

Flag set on the cross sections that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the cross sections in model m flagged with f:

```
CrossSection.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the cross section is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if cross section c is blanked:

```
if (c.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse cross section c:

```
c.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the cross section.

Arguments

- **flag** (Flag)

Flag to clear on the cross section

Returns

No return value

Example

To clear flag f for cross section c:

```
c.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the cross section. The target include of the copied cross section can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

CrossSection object

Return type

CrossSection

Example

To copy cross section c into cross section z:

```
var z = c.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a cross section

Arguments

- **Model** (Model)

Model that the cross section will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

CrossSection object (or null if not made).

Return type

CrossSection

Example

To start creating a cross section in model m:

```
var c = CrossSection.Create(m);
```

CreateAlongFeatureLine(Model*[Model]*, nid1*[integer]*, options*[object]*) [static]

Description

Creates a set of cross sections along a feature line and returns them as an array of CrossSection objects. Use Options.edge_angle to control the break angle for the feature line search within this function.

Arguments

- **Model** (Model)

Model that the cross_section will be created in

- **nid1** (integer)

ID of feature line starting node. The first cross section will be created at this Node's location.

- **options** (object)

Additional arguments for controlling how the cross sections are created.

Object has the following properties:

Name	Type	Description
autosize (optional)	boolean	true if cross sections are to be autosized. lenl and lenm will be ignored if set.
autosize_pct (optional)	real	If autosize is true, cross sections will be scaled by this additional percentage after being autosized. (default: same as value specified by the preference primer*cross_section_auto_size_percent)
direction (optional)	integer	Direction along feature line to create cross sections. +1 for positive direction (default), -1 for negative direction. Ignored for two node method (nid2 set).
lenl	real	Side length of plane along L vector - non-circular cross sections only.
lenm	real	Side length of plane along M vector - non-circular cross sections only.
nid2 (optional)	integer	ID of feature line ending node, can be the same as nid1. If provided, cross sections will be created between the nodes nid1 and nid2 along the shortest feature line path.
num	integer	Number of cross sections to be created. Required for single node method. For two node method either num or pitch must be specified. For two node method, the last cross section will be created at the location of node nid2 (provided nid1 != nid2).
pitch	real	Separation between adjacent cross sections. Required for single node method. For two node method either num or pitch must be specified.
psid (optional)	integer	ID of part set.
radius (optional)	real	Radius of circular cross section.
vis_only (optional)	boolean	If true, and autosize is true, autosizing will ignore non-visible cut elements, false otherwise (default: true).

Returns

Array of CrossSection objects (or null if not made). Depending on the geometry of the model and the node provided for nid1, the array may contain less CrossSection objects than requested for the single node method.

Return type

Array

Example

To create 10 autosized cross sections, separated by 15.0, in the positive direction, starting at Node 1:

```
var options = { num: 10, pitch: 15.0, direction: 1, autosize: true };
      var xsec_array = CrossSection.CreateAlongFeatureLine(model, 1, options);
```

To create 5 circular cross sections, of radius 25.0, around an entire perimeter edge (starting at Node 2):

```
var options = { num: 5, nid2: 2, radius: 25.0 };
      var xsec_array = CrossSection.CreateAlongFeatureLine(model, 2, options);
```

To create a set of cross sections separated by 10.0, of size 50.0 x 50.0, between nodes 3 and 4 (starting at Node 3):

```
var options = { pitch: 10.0, lenm: 50.0, lenl: 50.0, nid2: 4 };
      var xsec_array = CrossSection.CreateAlongFeatureLine(model, 3, options);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a cross section.

Arguments

- **Comment** (Comment)

Comment that will be detached from the cross section

Returns

No return value

Example

To detach comment c from the cross section c:

```
c.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit cross section c:

```
c.Edit();
```

ElemCut(Shell label*[integer]*)

Description

Returns coordinates of the intersections between a shell and a database cross section. Note, ElemCut on the Shell class may be quicker

Arguments

- **Shell label** (integer)

The label of the shell.

Returns

An array containing the x1,y1,z1,x2,y2,z2 coordinates of the cut line, or NULL if it does not cut. Note this function does not check that the shell is in the cross section definition (part set)

Return type

Array

Example

To get the cut line coordinates between database cross section x and shell 300:

```
var data = x.ElemCut(300)
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for cross section with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The cross section object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for cross section c:

```
c.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for cross section c:

```
c.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for cross section. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for cross section c:

```
c.Error("My custom error");
```

ExtractColour()

Description

Extracts the **actual** colour used for cross section.
By default in PRIMER many entities such as elements get their colour automatically from the part that they are in. PRIMER cycles through 13 default colours based on the label of the entity. In this case the cross section colour property will return the value Colour.PART instead of the actual colour. This method will return the actual colour which is used for drawing the cross section.

Arguments

No arguments

Returns

colour value (integer)

Return type

Number

Example

To return the colour used for drawing cross section c:

```
var colour = c.ExtractColour();
```

First(Model*[Model]*) [static]

Description

Returns the first cross section in the model.

Arguments

- **Model** (Model)

Model to get first cross section in

Returns

CrossSection object (or null if there are no cross sections in the model).

Return type

CrossSection

Example

To get the first cross section in model m:

```
var c = CrossSection.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free cross section label in the model. Also see CrossSection.LastFreeLabel(), CrossSection.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free cross section label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

CrossSection label.

Return type

Number

Example

To get the first free cross section label in model m:

```
var label = CrossSection.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the cross sections in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all cross sections will be flagged in

- **flag** (Flag)

Flag to set on the cross sections

Returns

No return value

Example

To flag all of the cross sections with flag f in model m:

```
CrossSection.FlagAll(m, f);
```

FlagCut(Flag*[Flag]*)

Description

Flags every element (solid,shell,tshell,beam) cut by the cross section. Note this function does not check that the element is in the cross section definition (part set)

Arguments

- **Flag** (Flag)

Flag bit.

Returns

Boolean.

Return type

Boolean

Example

To find elements cut by database cross section x:

```
x.FlagCut(flag)
```

Flagged(flag*[Flag]*)

Description

Checks if the cross section is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the cross section

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if cross section c has flag f set on it:

```
if (c.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each cross section in the model.
Note that ForEach has been designed to make looping over cross sections as fast as possible and so has some limitations.
Firstly, a single temporary CrossSection object is created and on each function call it is updated with the current cross section data. This means that you should not try to store the CrossSection object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new cross sections inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all cross sections are in

- **func** (function)

Function to call for each cross section

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the cross sections in model m:

```
CrossSection.ForEach(m, test);
function test(c)
{
// c is CrossSection object
}
```

To call function test for all of the cross sections in model m with optional object:

```
var data = { x:0, y:0 };
CrossSection.ForEach(m, test, data);
function test(c, extra)
{
// c is CrossSection object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of CrossSection objects or properties for all of the cross sections in a model in PRIMER. If the optional property argument is not given then an array of CrossSection objects is returned. If the property argument is given, that property value for each cross section is returned in the array instead of a CrossSection object

Arguments

- **Model** (Model)

Model to get cross sections from

- **property (optional)** (string)

Name for property to get for all cross sections in the model

Returns

Array of CrossSection objects or properties

Return type

Array

Example

To make an array of CrossSection objects for all of the cross sections in model m:

```
var a = CrossSection.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each cross section in model m:

```
var a = CrossSection.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a cross section.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the cross section c:

```
var comm_array = c.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of CrossSection objects for all of the flagged cross sections in a model in PRIMER If the optional property argument is not given then an array of CrossSection objects is returned. If the property argument is given, then that property value for each cross section is returned in the array instead of a CrossSection object

Arguments

- **Model** (Model)

Model to get cross sections from

- **flag** (Flag)

Flag set on the cross sections that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged cross sections in the model

Returns

Array of CrossSection objects or properties

Return type

Array

Example

To make an array of CrossSection objects for all of the cross sections in model m flagged with f:

```
var c = CrossSection.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the cross sections in model m flagged with f:

```
var a = CrossSection.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the CrossSection object for a cross section ID.

Arguments

- **Model** (Model)

Model to find the cross section in

- **number** (integer)

number of the cross section you want the CrossSection object for

Returns

CrossSection object (or null if cross section does not exist).

Return type

CrossSection

Example

To get the CrossSection object for cross section 100 in model m

```
var c = CrossSection.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a CrossSection property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the CrossSection.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

cross section property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if CrossSection property c.example is a parameter:

```
Options.property_parameter_names = true;
if (c.GetParameter(c.example) ) do_something...
Options.property_parameter_names = false;
```

To check if CrossSection property c.example is a parameter by using the GetParameter method:

```
if (c.ViewParameters().GetParameter(c.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this cross_section (*DATABASE_CROSS_SECTION). **Note that a carriage return is not added**. See also CrossSection.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for cross_section c:

```
var key = c.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the cross_section. **Note that a carriage return is not added**. See also CrossSection.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for cross_section c:

```
var cards = c.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last cross section in the model.

Arguments

- **Model** (Model)

Model to get last cross section in

Returns

CrossSection object (or null if there are no cross sections in the model).

Return type

CrossSection

Example

To get the last cross section in model m:

```
var c = CrossSection.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free cross section label in the model. Also see CrossSection.FirstFreeLabel(), CrossSection.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free cross section label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

CrossSection label.

Return type

Number

Example

To get the last free cross section label in model m:

```
var label = CrossSection.LastFreeLabel(m);
```

Next()

Description

Returns the next cross section in the model.

Arguments

No arguments

Returns

CrossSection object (or null if there are no more cross sections in the model).

Return type

CrossSection

Example

To get the cross section in model m after cross section c:

```
var c = c.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) cross section label in the model. Also see CrossSection.FirstFreeLabel(), CrossSection.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free cross section label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

CrossSection label.

Return type

Number

Example

To get the next free cross section label in model m:

```
var label = CrossSection.NextFreeLabel(m);
```

PartCut(Part label*[integer]*, Flag (optional)*[Flag]*)

Description

Returns true if cross section is cutting the part, false otherwise. If option flag is active, will flag every element of the part cut by the cross section. Note this function does not check that the part is in the cross section definition (part set)

Arguments

- **Part label** (integer)

The label of the part.

- **Flag (optional)** (Flag)

Optional Flag to flag the element which are cut by the cross section.

Returns

Boolean.

Return type

Boolean

Example

To know if a database cross section x cuts part 300:

```
x.PartCut(300)
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a cross section.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only cross sections from that model can be picked. If the argument is a Flag then only cross sections that are flagged with *limit* can be selected. If omitted, or null, any cross sections from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

CrossSection object (or null if not picked)

Return type

CrossSection

Example

To pick a cross section from model m giving the prompt ’Pick cross section from screen’:

```
var c = CrossSection.Pick(’Pick cross section from screen’, m);
```

Previous()

Description

Returns the previous cross section in the model.

Arguments

No arguments

Returns

CrossSection object (or null if there are no more cross sections in the model).

Return type

CrossSection

Example

To get the cross section in model m before cross section c:

```
var c = c.Previous();
```

Properties()

Description

Returns an object which describe various cross section properties

Arguments

No arguments

Returns

Object with the following properties:

Name	Type	Description
area	real	Area of material sliced by the cut section
first_yield_axial	real	First yield axial force
first_yield_mxx	real	First yield bending moment (Mxx)
first_yield_myy	real	First yield bending moment (Myy)
fully_plastic_mxx	real	Fully plastic bending moment (Mxx)
fully_plastic_myy	real	Fully plastic bending moment (Myy)
fully_plastic_xf	real	X component of equal force axis
fully_plastic_xf_g	real	X component of equal force axis calculated in global coordinates
fully_plastic_yf	real	Fully plastic axial force
fully_plastic_yf_g	real	Y component of equal force axis calculated in global coordinates

fully_plastic_zf_g	real	Z component of equal force axis calculated in global coordinates
iuu	real	Iuu principal second moments (UU - major)
ivv	real	Ivv principal second moments (VV - minor)
ixx	real	Ixx component of second moment of area
ixy	real	Ixy component of second moment of area
iyy	real	Iyy component of second moment of area
origin_x	real	X component of section origin
origin_y	real	Y component of section origin
origin_z	real	Z component of section origin
theta	real	Angle between Ixx and Iuu
x_comp_axis_x	real	X component of X-axis vector
x_comp_axis_y	real	X component of Y-axis vector
x_comp_axis_z	real	X component of Z-axis vector
xc	real	X component of centroid calculated from the first moment of area
xc_global	real	X component of centre of gravity calculated in global coordinates
xe	real	X component of equal area axis
xe_global	real	X component of equal area axis calculated in global coordinates
y_comp_axis_x	real	Y component of x-axis vector
y_comp_axis_y	real	Y component of Y-axis vector
y_comp_axis_z	real	Y component of Z-axis vector
yc	real	Y component of centroid calculated from the first moment of area
yc_global	real	Y component of centre of gravity calculated in global coordinates
ye	real	Y component of equal area axis
ye_global	real	Y component of equal area axis calculated in global coordinates
z_comp_axis_x	real	Z component of X-axis vector
z_comp_axis_y	real	Z component of Y-axis vector
z_comp_axis_z	real	Z component of Z-axis vector
zc_global	real	Z component of centre of gravity calculated in global coordinates
ze_global	real	Z component of equal area axis calculated in global coordinates
zxx	real	Plastic moduli Zxx
zyy	real	Plastic moduli Zyy

Return type

object

Example

To get the cross section properties for section c:

```
var properties = c.Properties();
    var originX  = properties.origin_x;
    var originY  = properties.origin_y;
    var Xc = properties.xc;
    var Ixx = properties.ixx;
    var Iuu = properties.iuu;
    var Theta = properties.theta;
    var plastic_Mxx = properties.fully_plastic_mxx;
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the cross sections in the model.

Arguments

- **Model** (Model)

Model that all cross sections will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the cross sections in model m, from 1000000:

```
CrossSection.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged cross sections in the model.

Arguments

- **Model** (Model)

Model that all the flagged cross sections will be renumbered in

- **flag** (Flag)

Flag set on the cross sections that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the cross sections in model m flagged with f, from 1000000:

```
CrossSection.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select cross sections using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting cross sections

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only cross sections from that model can be selected. If the argument is a Flag then only cross sections that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any cross sections can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of cross sections selected or null if menu cancelled

Return type

Number

Example

To select cross sections from model m, flagging those selected with flag f, giving the prompt 'Select cross sections':

```
CrossSection.Select(f, 'Select cross sections', m);
```

To select cross sections, flagging those selected with flag f but limiting selection to cross sections flagged with flag l, giving the prompt 'Select cross sections':

```
CrossSection.Select(f, 'Select cross sections', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the cross section.

Arguments

- **flag** (Flag)

Flag to set on the cross section

Returns

No return value

Example

To set flag f for cross section c:

```
c.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the cross section. The cross section will be sketched until you either call CrossSection.Unsketch(), CrossSection.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the cross section is sketched. If omitted redraw is true. If you want to sketch several cross sections and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch cross section c:

```
c.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged cross sections in the model. The cross sections will be sketched until you either call CrossSection.Unsketch(), CrossSection.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged cross sections will be sketched in

- **flag** (Flag)

Flag set on the cross sections that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the cross sections are sketched. If omitted redraw is true. If you want to sketch flagged cross sections several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all cross sections flagged with flag in model m:

```
CrossSection.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of cross sections in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing cross sections should be counted. If false or omitted referenced but undefined cross sections will also be included in the total.

Returns

number of cross sections

Return type

Number

Example

To get the total number of cross sections in model m:

```
var total = CrossSection.Total(m);
```

Unblank()

Description

Unblanks the cross section

Arguments

No arguments

Returns

No return value

Example

To unblank cross section c:

```
c.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the cross sections in the model.

Arguments

- **Model** (Model)

Model that all cross sections will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the cross sections in model m:

```
CrossSection.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged cross sections in the model.

Arguments

- **Model** (Model)

Model that the flagged cross sections will be unblanked in

- **flag** (Flag)

Flag set on the cross sections that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the cross sections in model m flagged with f:

```
CrossSection.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the cross sections in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all cross sections will be unset in

- **flag** (Flag)

Flag to unset on the cross sections

Returns

No return value

Example

To unset the flag f on all the cross sections in model m:

```
CrossSection.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the cross section.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the cross section is unsketched. If omitted redraw is true. If you want to unsketch several cross sections and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch cross section c:

```
c.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all cross sections.

Arguments

- **Model** (Model)

Model that all cross sections will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the cross sections are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all cross sections in model m:

```
CrossSection.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged cross sections in the model.

Arguments

- **Model** (Model)

Model that all cross sections will be unsketched in

- **flag** (Flag)

Flag set on the cross sections that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the cross sections are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all cross sections flagged with flag in model m:

```
CrossSection.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

CrossSection object.

Return type

CrossSection

Example

To check if CrossSection property c.example is a parameter by using the CrossSection.GetParameter() method:

```
if (c.ViewParameters().GetParameter(c.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for cross section with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The cross section object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for cross section c:

```
c.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for cross section c:

```
c.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for cross section. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for cross section c:

```
c.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this cross section.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for cross section c:

```
var xrefs = c.Xrefs();
```

---

toString()

Description

Creates a string containing the cross_section data in keyword format. Note that this contains the keyword header and the keyword cards. See also CrossSection.Keyword() and CrossSection.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for cross_section c in keyword format

```
var s = c.toString();
```

History class

The History class gives you access to database history cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, type (optional)*[constant]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, type (optional)*[constant]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, type*[constant]*, modal (optional)*[boolean]*)
- First(Model*[Model]*, type (optional)*[constant]*)
- FlagAll(Model*[Model]*, flag*[Flag]*, type (optional)*[constant]*)
- GetAll(Model*[Model]*, type (optional)*[constant]*)
- GetFromID(Model*[Model]*, database history number*[integer]*)
- Last(Model*[Model]*, type (optional)*[constant]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, type (optional)*[constant]*, redraw (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*, type (optional)*[constant]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, type (optional)*[constant]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*, type (optional)*[constant]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, type (optional)*[constant]*, redraw (optional)*[boolean]*)

Member functions

- Blanked()
- ClearFlag(flag*[Flag]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unsketch(redraw (optional)*[boolean]*)
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

History constants

Name	Description
History.ACOUSTIC	ACOUSTIC is *DATABASE_HISTORY_ACOUSTIC.
History.ALL_TYPES	All *DATABASE_HISTORY_ types.
History.BEAM	BEAM is *DATABASE_HISTORY_BEAM.
History.BEAM_SET	BEAM_SET is *DATABASE_HISTORY_BEAM_SET.
History.DISCRETE	DISCRETE is *DATABASE_HISTORY_DISCRETE.

History.DISCRETE_SET	DISCRETE_SET is *DATABASE_HISTORY_DISCRETE_SET.
History.NODE	NODE is *DATABASE_HISTORY_NODE.
History.NODE_SET	NODE_SET is *DATABASE_HISTORY_NODE_SET.
History.SEATBELT	SEATBELT is *DATABASE_HISTORY_SEATBELT.
History.SHELL	SHELL is *DATABASE_HISTORY_SHELL.
History.SHELL_SET	SHELL_SET is *DATABASE_HISTORY_SHELL_SET.
History.SOLID	SOLID is *DATABASE_HISTORY_SOLID.
History.SOLID_SET	SOLID_SET is *DATABASE_HISTORY_SOLID_SET.
History.SPH	SPH is *DATABASE_HISTORY_SPH.
History.SPH_SET	SPH_SET is *DATABASE_HISTORY_SPH_SET.
History.TSHELL	TSHELL is *DATABASE_HISTORY_TSHELL.
History.TSHELL_SET	TSHELL_SET is *DATABASE_HISTORY_TSHELL_SET.

History properties

Name	Type	Description
cid	integer	Coordinate system ID for _LOCAL
exists (read only)	logical	true if database history exists, false if referred to but not defined.
heading	string	Optional heading
hfo	integer	High frequency flag for _LOCAL
id	integer	ID of the item
include	integer	The Include file number that the database history is in.
local	logical	Turns _LOCAL on or off
model	integer	The Model number that the database history is in.
ref	integer	Output reference for _LOCAL
type (read only)	constant	The database history type. Can be History.ACOUSTIC or History.BEAM or History.BEAM_SET or History.DISCRETE or History.DISCRETE_SET or History.NODE or History.NODE_SET or History.SEATBELT or History.SHELL or History.SHELL_SET or History.SOLID or History.SOLID_SET or History.SPH or History.SPH_SET or History.TSHELL or History.TSHELL_SET.

Detailed Description

The History class allows you to create, modify, edit and manipulate database history cards. See the documentation below for more details.

Constructor

new History(Model*[Model]*, type*[constant]*, id*[integer]*, heading (optional)*[string]*)

Description

Create a new History object.

Arguments

- **Model** (Model)

Model that database history will be created in

- **type** (constant)

Entity type

- **id** (integer)

ID of the item

- **heading (optional)** (string)

Optional heading

Returns

History object

Return type

History

Example

To create a new Database history on NODE 500 called "test history":

```
var c = new History(m, History.NODE, 500, "test history");
```

Details of functions

BlankAll(Model*[Model]*, type (optional)*[constant]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the database histories in the model.

Arguments

- **Model** (Model)

Model that all database histories will be blanked in

- **type (optional)** (constant)

The database history type. Can be History.ACOUSTIC or History.BEAM or History.BEAM_SET or History.DISCRETE or History.DISCRETE_SET or History.NODE or History.NODE_SET or History.SEATBELT or History.SHELL or History.SHELL_SET or History.SOLID or History.SOLID_SET or History.SPH or History.SPH_SET or History.TSHELL or History.TSHELL_SET or History.ALL_TYPES. If omitted, applied to all database history types.

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the database histories in model m:

```
History.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, type (optional)*[constant]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged database histories in the model.

Arguments

- **Model** (Model)

Model that all the flagged database histories will be blanked in

- **flag** (Flag)

Flag set on the database histories that you want to blank

- **type (optional)** (constant)

The database history type. Can be History.ACOUSTIC or History.BEAM or History.BEAM_SET or History.DISCRETE or History.DISCRETE_SET or History.NODE or History.NODE_SET or History.SEATBELT or History.SHELL or History.SHELL_SET or History.SOLID or History.SOLID_SET or History.SPH or History.SPH_SET or History.TSHELL or History.TSHELL_SET or History.ALL_TYPES. If omitted, applied to all database history types.

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the database histories in model m flagged with f:

```
History.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the database history is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if database history c is blanked:

```
if (c.Blanked() ) do_something...
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the database history.

Arguments

- **flag** (Flag)

Flag to clear on the database history

Returns

No return value

Example

To clear flag f for database history c:

```
c.ClearFlag(f);
```

Create(Model*[Model]*, type*[constant]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a database history.

Arguments

- **Model** (Model)

Model that the database history will be created in

- **type** (constant)

The database history type. Can be History.ACOUSTIC or History.BEAM or History.BEAM_SET or History.DISCRETE or History.DISCRETE_SET or History.NODE or History.NODE_SET or History.SEATBELT or History.SHELL or History.SHELL_SET or History.SOLID or History.SOLID_SET or History.SPH or History.SPH_SET or History.TSHELL or History.TSHELL_SET.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

History object (or null if not made)

Return type

History

Example

To start creating a history in model m:

```
var c = History.Create(m);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel to edit the database history.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

No return value

Example

To edit database history c:

```
c.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for a database history with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The database history object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for database history h:

```
h.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for database history h:

```
h.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for a database history. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for database history h:

```
h.Error("My custom error");
```

First(Model*[Model]*, type (optional)*[constant]*) [static]

Description

Returns the first database history in the model.

Arguments

- **Model** (Model)

Model to get first database history in

- **type (optional)** (constant)

The database history type. Can be History.ACOUSTIC or History.BEAM or History.BEAM_SET or History.DISCRETE or History.DISCRETE_SET or History.NODE or History.NODE_SET or History.SEATBELT or History.SHELL or History.SHELL_SET or History.SOLID or History.SOLID_SET or History.SPH or History.SPH_SET or History.TSHELL or History.TSHELL_SET or History.ALL_TYPES. If omitted, applied to all database history types.

Returns

History object (or null if there are no database histories in the model).

Return type

History

Example

To get the first database history in model m:

```
var history = History.First(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*, type (optional)*[constant]*) [static]

Description

Flags all of the database histories in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all database histories will be flagged in

- **flag** (Flag)

Flag to set on the database histories

- **type (optional)** (constant)

The database history type. Can be History.ACOUSTIC or History.BEAM or History.BEAM_SET or History.DISCRETE or History.DISCRETE_SET or History.NODE or History.NODE_SET or History.SEATBELT or History.SHELL or History.SHELL_SET or History.SOLID or History.SOLID_SET or History.SPH or History.SPH_SET or History.TSHELL or History.TSHELL_SET or History.ALL_TYPES. If omitted, applied to all database history types.

Returns

No return value

Example

To flag all of the database histories with flag f in model m:

```
History.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the database history is flagged or not.

Arguments

- **flag** (Flag)

Flag to clear on the database history

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if database history c has flag f set on it:

```
if (c.Flagged(f) ) do_something...
```

GetAll(Model*[Model]*, type (optional)*[constant]*) [static]

Description

Returns an array of History objects for all of the database histories in a models in PRIMER

Arguments

- **Model** (Model)

Model to get database histories from

- **type (optional)** (constant)

The database history type. Can be History.ACOUSTIC or History.BEAM or History.BEAM_SET or History.DISCRETE or History.DISCRETE_SET or History.NODE or History.NODE_SET or History.SEATBELT or History.SHELL or History.SHELL_SET or History.SOLID or History.SOLID_SET or History.SPH or History.SPH_SET or History.TSHELL or History.TSHELL_SET or History.ALL_TYPES. If omitted, applied to all database history types.

Returns

Array of History objects

Return type

Array

Example

To make an array of History objects for all of the database histories in model m

```
var database history = History.GetAll(m);
```

GetFromID(Model*[Model]*, database history number*[integer]*) [static]

Description

Returns the History object for a database history ID.

Arguments

- **Model** (Model)

Model to find the database history in

- **database history number** (integer)

number of the database history you want the History object for

Returns

History object (or null if database history does not exist).

Return type

History

Example

To get the History object for database history 100 in model m

```
var database history = History.GetFromID(m, 100);
```

Keyword()

Description

Returns the keyword for this database history (*DATABASE_HISTORY). **Note that a carriage return is not added**. See also History.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for database history c:

```
var key = c.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the database history. **Note that a carriage return is not added**. See also History.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for database history c:

```
var cards = c.KeywordCards();
```

Last(Model*[Model]*, type (optional)*[constant]*) [static]

Description

Returns the last database history in the model.

Arguments

- **Model** (Model)

Model to get last database history in

- **type (optional)** (constant)

The database history type. Can be History.ACOUSTIC or History.BEAM or History.BEAM_SET or History.DISCRETE or History.DISCRETE_SET or History.NODE or History.NODE_SET or History.SEATBELT or History.SHELL or History.SHELL_SET or History.SOLID or History.SOLID_SET or History.SPH or History.SPH_

SET or History.TSHELL or History.TSHELL_SET or History.ALL_TYPES. If omitted, applied to all database history types.

Returns

History object (or null if there are no database histories in the model).

Return type

History

Example

To get the last database history in model m:

```
var database history = History.Last(m);
```

Next()

Description

Returns the next database history in the model.

Arguments

No arguments

Returns

History object (or null if there are no more database histories in the model).

Return type

History

Example

To get the database history in model m after database history c:

```
var database history = c.Next();
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to pick a database history.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only database histories from that model can be picked. If the argument is a Flag then only database histories that are flagged with *limit* can be selected. If omitted, or null, any database histories from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

Returns

History object (or null if not picked)

Return type

History

Example

To pick a database history from model m giving the prompt 'Pick database history from screen':

```
var database history = History.Pick('Pick database history from screen', m);
```

Previous()

Description

Returns the previous database history in the model.

Arguments

No arguments

Returns

History object (or null if there are no more database histories in the model).

Return type

History

Example

To get the database history in model m before this one:

```
var history = history.Previous();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select database histories using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting database histories

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only database histories from that model can be selected. If the argument is a Flag then only database histories that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any database histories from any model can be selected.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of items selected or null if menu cancelled

Return type

Number

Example

To select database histories from model m, flagging those selected which flag f, giving the prompt 'Select database histories':

```
History.Select(f, 'Select database histories', m);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the database history.

Arguments

- **flag** (Flag)

Flag to set on the database history

Returns

No return value

Example

To set flag f for database history c:

```
c.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the database history. The database history will be sketched until you either call History.Unsketch(), History.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the database history is sketched. If omitted redraw is true. If you want to sketch several database histories and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch database history c:

```
c.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, type (optional)*[constant]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged database histories in the model.

Arguments

- **Model** (Model)

Model that all the flagged database histories will be sketched in

- **flag** (Flag)

Flag set on the database histories that you want to sketch

- **type (optional)** (constant)

The database history type. Can be History.ACOUSTIC or History.BEAM or History.BEAM_SET or History.DISCRETE or History.DISCRETE_SET or History.NODE or History.NODE_SET or History.SEATBELT or History.SHELL or History.SHELL_SET or History.SOLID or History.SOLID_SET or History.SPH or History.SPH_SET or History.TSHELL or History.TSHELL_SET or History.ALL_TYPES. If omitted, applied to all database history types.

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is true. If you want to do several (un)sketches and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To sketch all of the database histories of type SHELL_SET in model m flagged with f:

```
History.SketchFlagged(m, f, History.SHELL_SET);
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*, type (optional)*[constant]*) [static]

Description

Unblanks all of the database histories in the model.

Arguments

- **Model** (Model)

Model that all database histories will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

- **type (optional)** (constant)

The database history type. Can be History.ACOUSTIC or History.BEAM or History.BEAM_SET or History.DISCRETE or History.DISCRETE_SET or History.NODE or History.NODE_SET or History.SEATBELT or History.SHELL or History.SHELL_SET or History.SOLID or History.SOLID_SET or History.SPH or History.SPH_SET or History.TSHELL or History.TSHELL_SET or History.ALL_TYPES. If omitted, applied to all database history types.

Returns

No return value

Example

To unblank all of the database histories in model m:

```
History.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, type (optional)*[constant]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged database histories in the model.

Arguments

- **Model** (Model)

Model that the flagged database histories will be unblanked in

- **flag** (Flag)

Flag set on the database histories that you want to unblank

- **type (optional)** (constant)

The database history type. Can be History.ACOUSTIC or History.BEAM or History.BEAM_SET or History.DISCRETE or History.DISCRETE_SET or History.NODE or History.NODE_SET or History.SEATBELT or History.SHELL or History.SHELL_SET or History.SOLID or History.SOLID_SET or History.SPH or History.SPH_SET or History.TSHELL or History.TSHELL_SET or History.ALL_TYPES. If omitted, applied to all database history types.

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the database histories in model m flagged with f:

```
History.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*, type (optional)*[constant]*) [static]

Description

Unsets a defined flag on all of the database histories in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all database histories will be unset in

- **flag** (Flag)

Flag to unset on the database histories

- **type (optional)** (constant)

The database history type. Can be History.ACOUSTIC or History.BEAM or History.BEAM_SET or History.DISCRETE or History.DISCRETE_SET or History.NODE or History.NODE_SET or History.SEATBELT or History.SHELL or History.SHELL_SET or History.SOLID or History.SOLID_SET or History.SPH or History.SPH_SET or History.TSHELL or History.TSHELL_SET or History.ALL_TYPES. If omitted, applied to all database history types.

Returns

No return value

Example

To unset the flag f on all the database histories in model m:

```
History.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the database history.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the database history is unsketched. If omitted redraw is true. If you want to unsketch several database histories and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch database history c:

```
c.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all database histories.

Arguments

- **Model** (Model)

Model that all database histories will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the database histories are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all database histories in model m:

```
History.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, type (optional)*[constant]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged database histories.

Arguments

- **Model** (Model)

Model that all database histories will be unblanked in

- **flag** (Flag)

Flag set on the database histories that you want to sketch

- **type (optional)** (constant)

The database history type. Can be History.ACOUSTIC or History.BEAM or History.BEAM_SET or History.DISCRETE or History.DISCRETE_SET or History.NODE or History.NODE_SET or History.SEATBELT or History.SHELL or History.SHELL_SET or History.SOLID or History.SOLID_SET or History.SPH or History.SPH_SET or History.TSHELL or History.TSHELL_SET or History.ALL_TYPES. If omitted, applied to all database history types.

- **redraw (optional)** (boolean)

If model should be redrawn or not after the database histories are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all of the database histories in model m flagged with f:

```
History.UnsketchFlagged(m, f);
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for a database history with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The database history object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for database history h:

```
h.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for database history h:

```
h.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for a database history. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for database history h:

```
h.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this database history.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for this database history:

```
var xrefs = c.Xrefs();
```

---

toString()

Description

Creates a string containing the database history data in keyword format. Note that this contains the keyword header and the keyword cards. See also History.Keyword() and History.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for database history c in keyword format

```
var s = c.toString();
```

NodalForceGroup (Nfgr) class

The NodalForceGroup class gives you access to database nodal force group cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- First(Model*[Model]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

NodalForceGroup properties

Name	Type	Description

cid	integer	Coordinate System ID.
exists (read only)	logical	true if nodal force group exists, false if referred to but not defined.
id	integer	Database Nodal Force Group number (identical to label).
include	integer	The Include file number that the nodal force group is in.
label	integer	Database Nodal Force Group number.
model (read only)	integer	The Model number that the nodal force group is in.
nsid	integer	Set Node Set ID.

Detailed Description

The NodalForceGroup class allows you to create, modify, edit and manipulate nodal force group cards. See the documentation below for more details.

For convenience "Nfgr" can also be used as the class name instead of "NodalForceGroup".

Constructor

new NodalForceGroup(Model*[Model]*, nsid*[integer]*, cid (optional)*[integer]*)

Description

Create a new NodalForceGroup object.

Arguments

- **Model** (Model)

Model that nodal force group will be created in

- **nsid** (integer)

Set Node Set ID.

- **cid (optional)** (integer)

Coordinate System ID.

Returns

NodalForceGroup object

Return type

NodalForceGroup

Example

To create a new nodal force group in model m with nsid 100:

```
var nfg = new NodalForceGroup(m, 100);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a nodal force group.

Arguments

- **Comment** (Comment)

Comment that will be attached to the nodal force group

Returns

No return value

Example

To associate comment c to the nodal force group nfg:

```
nfg.AssociateComment(c);
```

Blank()

Description

Blanks the nodal force group

Arguments

No arguments

Returns

No return value

Example

To blank nodal force group nfg:

```
nfg.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the nodal force groups in the model.

Arguments

- **Model** (Model)

Model that all nodal force groups will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the nodal force groups in model m:

```
NodalForceGroup.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged nodal force groups in the model.

Arguments

- **Model** (Model)

Model that all the flagged nodal force groups will be blanked in

- **flag** (Flag)

Flag set on the nodal force groups that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the nodal force groups in model m flagged with f:

```
NodalForceGroup.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the nodal force group is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if nodal force group nfg is blanked:

```
if (nfg.Blanked() ) do_something...
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the nodal force group.

Arguments

- **flag** (Flag)

Flag to clear on the nodal force group

Returns

No return value

Example

To clear flag f for nodal force group nfg:

```
nfg.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the nodal force group. The target include of the copied nodal force group can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

NodalForceGroup object

Return type

NodalForceGroup

Example

To copy nodal force group nfg into nodal force group z:

```
var z = nfg.Copy();
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a nodal force group.

Arguments

- **Comment** (Comment)

Comment that will be detached from the nodal force group

Returns

No return value

Example

To detach comment c from the nodal force group nfg:

```
nfg.DetachComment(c);
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for nodal force group with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The nodal force group object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for nodal force group nfg:

```
nfg.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for nodal force group nfg:

```
nfg.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for nodal force group. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for nodal force group nfg:

```
nfg.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first nodal force group in the model.

Arguments

- **Model** (Model)

Model to get first nodal force group in

Returns

NodalForceGroup object (or null if there are no nodal force groups in the model).

Return type

NodalForceGroup

Example

To get the first nodal force group in model m:

```
var nfg = NodalForceGroup.First(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the nodal force groups in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all nodal force groups will be flagged in

- **flag** (Flag)

Flag to set on the nodal force groups

Returns

No return value

Example

To flag all of the nodal force groups with flag f in model m:

```
NodalForceGroup.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the nodal force group is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the nodal force group

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if nodal force group nfg has flag f set on it:

```
if (nfg.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each nodal force group in the model.
Note that ForEach has been designed to make looping over nodal force groups as fast as possible and so has some limitations.
Firstly, a single temporary NodalForceGroup object is created and on each function call it is updated with the current nodal force group data. This means that you should not try to store the NodalForceGroup object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new nodal force groups inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all nodal force groups are in

- **func** (function)

Function to call for each nodal force group

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the nodal force groups in model m:

```
NodalForceGroup.ForEach(m, test);
function test(nfg)
{
// nfg is NodalForceGroup object
}
```

To call function test for all of the nodal force groups in model m with optional object:

```
var data = { x:0, y:0 };
NodalForceGroup.ForEach(m, test, data);
function test(nfg, extra)
{
// nfg is NodalForceGroup object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of NodalForceGroup objects or properties for all of the nodal force groups in a model in PRIMER. If the optional property argument is not given then an array of NodalForceGroup objects is returned. If the property argument is given, that property value for each nodal force group is returned in the array instead of a NodalForceGroup object

Arguments

- **Model** (Model)

Model to get nodal force groups from

- **property (optional)** (string)

Name for property to get for all nodal force groups in the model

Returns

Array of NodalForceGroup objects or properties

Return type

Array

Example

To make an array of NodalForceGroup objects for all of the nodal force groups in model m:

```
var a = NodalForceGroup.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each nodal force group in model m:

```
var a = NodalForceGroup.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a nodal force group.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the nodal force group nfg:

```
var comm_array = nfg.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of NodalForceGroup objects for all of the flagged nodal force groups in a model in PRIMER If the optional property argument is not given then an array of NodalForceGroup objects is returned. If the property argument is given, then that property value for each nodal force group is returned in the array instead of a NodalForceGroup object

Arguments

- **Model** (Model)

Model to get nodal force groups from

- **flag** (Flag)

Flag set on the nodal force groups that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged nodal force groups in the model

Returns

Array of NodalForceGroup objects or properties

Return type

Array

Example

To make an array of NodalForceGroup objects for all of the nodal force groups in model m flagged with f:

```
var nfg = NodalForceGroup.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the nodal force groups in model m flagged with f:

```
var a = NodalForceGroup.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the NodalForceGroup object for a nodal force group ID.

Arguments

- **Model** (Model)

Model to find the nodal force group in

- **number** (integer)

number of the nodal force group you want the NodalForceGroup object for

Returns

NodalForceGroup object (or null if nodal force group does not exist).

Return type

NodalForceGroup

Example

To get the NodalForceGroup object for nodal force group 100 in model m

```
var nfg = NodalForceGroup.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a NodalForceGroup property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the NodalForceGroup.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

nodal force group property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if NodalForceGroup property nfg.example is a parameter:

```
Options.property_parameter_names = true;
if (nfg.GetParameter(nfg.example) ) do_something...
Options.property_parameter_names = false;
```

To check if NodalForceGroup property nfg.example is a parameter by using the GetParameter method:

```
if (nfg.ViewParameters().GetParameter(nfg.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this nodal force group. **Note that a carriage return is not added**. See also NodalForceGroup.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for nodal force group nfg:

```
var key = nfg.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the nodal force group. **Note that a carriage return is not added**. See also NodalForceGroup.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for nodal force group nfg:

```
var cards = nfg.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last nodal force group in the model.

Arguments

- **Model** (Model)

Model to get last nodal force group in

Returns

NodalForceGroup object (or null if there are no nodal force groups in the model).

Return type

NodalForceGroup

Example

To get the last nodal force group in model m:

```
var nfg = NodalForceGroup.Last(m);
```

Next()

Description

Returns the next nodal force group in the model.

Arguments

No arguments

Returns

NodalForceGroup object (or null if there are no more nodal force groups in the model).

Return type

NodalForceGroup

Example

To get the nodal force group in model m after nodal force group nfg:

```
var nfg = nfg.Next();
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a nodal force group.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only nodal force groups from that model can be picked. If the argument is a Flag then only nodal force groups that are flagged with *limit* can be selected. If omitted, or null, any nodal force groups from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

NodalForceGroup object (or null if not picked)

Return type

NodalForceGroup

Example

To pick a nodal force group from model m giving the prompt 'Pick nodal force group from screen':

```
var nfg = NodalForceGroup.Pick('Pick nodal force group from screen', m);
```

Previous()

Description

Returns the previous nodal force group in the model.

Arguments

No arguments

Returns

NodalForceGroup object (or null if there are no more nodal force groups in the model).

Return type

NodalForceGroup

Example

To get the nodal force group in model m before nodal force group nfg:

```
var nfg = nfg.Previous();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select nodal force groups using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting nodal force groups

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only nodal force groups from that model can be selected. If the argument is a Flag then only nodal force groups that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any nodal force groups can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of nodal force groups selected or null if menu cancelled

Return type

Number

Example

To select nodal force groups from model m, flagging those selected with flag f, giving the prompt 'Select nodal force groups':

```
NodalForceGroup.Select(f, 'Select nodal force groups', m);
```

To select nodal force groups, flagging those selected with flag f but limiting selection to nodal force groups flagged with flag l, giving the prompt 'Select nodal force groups':

```
NodalForceGroup.Select(f, 'Select nodal force groups', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the nodal force group.

Arguments

- **flag** (Flag)

Flag to set on the nodal force group

Returns

No return value

Example

To set flag f for nodal force group nfg:

```
nfg.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the nodal force group. The nodal force group will be sketched until you either call NodalForceGroup.Unsketch(), NodalForceGroup.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the nodal force group is sketched. If omitted redraw is true. If you want to sketch several nodal force groups and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch nodal force group nfg:

```
nfg.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged nodal force groups in the model. The nodal force groups will be sketched until you either call NodalForceGroup.Unsketch(), NodalForceGroup.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged nodal force groups will be sketched in

- **flag** (Flag)

Flag set on the nodal force groups that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the nodal force groups are sketched. If omitted redraw is true. If you want to sketch flagged nodal force groups several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all nodal force groups flagged with flag in model m:

```
NodalForceGroup.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of nodal force groups in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing nodal force groups should be counted. If false or omitted referenced but undefined nodal force groups will also be included in the total.

Returns

number of nodal force groups

Return type

Number

Example

To get the total number of nodal force groups in model m:

```
var total = NodalForceGroup.Total(m);
```

Unblank()

Description

Unblanks the nodal force group

Arguments

No arguments

Returns

No return value

Example

To unblank nodal force group nfg:

```
nfg.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the nodal force groups in the model.

Arguments

- **Model** (Model)

Model that all nodal force groups will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the nodal force groups in model m:

```
NodalForceGroup.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged nodal force groups in the model.

Arguments

- **Model** (Model)

Model that the flagged nodal force groups will be unblanked in

- **flag** (Flag)

Flag set on the nodal force groups that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the nodal force groups in model m flagged with f:

```
NodalForceGroup.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the nodal force groups in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all nodal force groups will be unset in

- **flag** (Flag)

Flag to unset on the nodal force groups

Returns

No return value

Example

To unset the flag f on all the nodal force groups in model m:

```
NodalForceGroup.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the nodal force group.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the nodal force group is unsketched. If omitted redraw is true. If you want to unsketch several nodal force groups and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch nodal force group nfg:

```
nfg.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all nodal force groups.

Arguments

- **Model** (Model)

Model that all nodal force groups will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the nodal force groups are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all nodal force groups in model m:

```
NodalForceGroup.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged nodal force groups in the model.

Arguments

- **Model** (Model)

Model that all nodal force groups will be unsketched in

- **flag** (Flag)

Flag set on the nodal force groups that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the nodal force groups are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all nodal force groups flagged with flag in model m:

```
NodalForceGroup.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

NodalForceGroup object.

Return type

NodalForceGroup

Example

To check if NodalForceGroup property nfg.example is a parameter by using the NodalForceGroup.GetParameter() method:

```
if (nfg.ViewParameters().GetParameter(nfg.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for nodal force group with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The nodal force group object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for nodal force group nfg:

```
nfg.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for nodal force group nfg:

```
nfg.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for nodal force group. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for nodal force group nfg:

```
nfg.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this nodal force group.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for nodal force group nfg:

```
var xrefs = nfg.Xrefs();
```

---

toString()

Description

Creates a string containing the nodal force group data in keyword format. Note that this contains the keyword header and the keyword cards. See also NodalForceGroup.Keyword() and NodalForceGroup.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for nodal force group n in keyword format

```
var s = n.toString();
```

Box class

The Box class gives you access to define box cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]

- Xrefs()
- toString()

Box constants

Name	Description
Box.BOX	Box is *DEFINE_BOX.
Box.BOX_ADAPTIVE	Box is *DEFINE_BOX_ADAPTIVE.
Box.BOX_COARSEN	Box is *DEFINE_BOX_COARSEN.
Box.BOX_DRAWBEAD	Box is *DEFINE_BOX_DRAWBEAD.
Box.BOX_SPH	Box is *DEFINE_BOX_SPH.

Box properties

Name	Type	Description
bid	integer	Box number. Also see the label property which is an alternative name for this.
brmax	real	Maximum mesh size in 3D tetrahedron adaptivity
brmin	real	Minimum mesh size in 3D tetrahedron adaptivity
cid	integer	Optional coordinate system ID for tubular drawbead
cx	real	X coordinate of offset vector to local origin
cy	real	Y coordinate of offset vector to local origin
cz	real	Z coordinate of offset vector to local origin
exists (read only)	logical	true if box exists, false if referred to but not defined.
heading	string	Box heading
idir	integer	Direction of tooling movement. 1: x-direction, 2: y-direction, 3: z-direction
iflag	integer	Element protection flag. 0: elements inside, 1: elements outside box cannot be coarsened.
include	integer	The Include file number that the box is in.
label	integer	Box number. Also see the bid property which is an alternative name for this.
lcid	integer	Load curve ID to describe motion value versus time
level	integer	Maximum number of refinement levels for elements contained in box
lidx	integer	Box movement in global X axis or by node. The ndid property is an alternative name for this.
lidy	integer	Box movement in global Y axis
lidz	integer	Box movement in global Z axis
local	logical	Turns _LOCAL on or off
model (read only)	integer	The Model number that the box is in.
ndid	integer	Box movement in global X axis or by node. The lidx property is an alternative name for this.
nid	integer	Referential nodal ID for vd = 2
option	constant	The box option. Can be Box.BOX, Box.BOX_ADAPTIVE, Box.BOX_COARSEN, Box.BOX_DRAWBEAD or Box.BOX_SPH.

pid_adaptive	integer	Part ID for Box.BOX_ADAPTIVE option
pid_drawbead	integer	Part ID of blank for Box.BOX_DRAWBEAD option
radius	real	Radius of tube centered around draw bead
sid	integer	Part set, part or node set defining the nodal points along draw bead
stype	integer	Set type for stype. 2: part set ID, 3: part ID, 4: node set ID
vd	integer	Velocity/Displacement flag. 0: velocity, 1: displacement, 2: referential node
vid	integer	Vector ID of DOF
xmn	real	Minimum X coordinate
xmx	real	Maximum X coordinate
xv	real	Local V vector X coordinate
xx	real	Local X vector X coordinate
ymn	real	Minimum Y coordinate
ymx	real	Maximum Y coordinate
yv	real	Local V vector Y coordinate
yx	real	Local X vector Y coordinate
zmn	real	Minimum Z coordinate
zmx	real	Maximum Z coordinate
zv	real	Local V vector Z coordinate
zx	real	Local X vector Z coordinate

Detailed Description

The Box class allows you to create, modify, edit and manipulate box cards. See the documentation below for more details.

Constructor

new Box(Model*[Model]*, bid*[integer]*, xmn*[real]*, xmx*[real]*, ymn*[real]*, ymx*[real]*, zmn*[real]*, zmx*[real]*, heading (optional)*[string]*)

Description

Create a new Box object.

Arguments

- **Model** (Model)

Model that box will be created in

- **bid** (integer)

Box number

- **xmn** (real)

Minimum X coordinate

- **xmx** (real)

Maximum X coordinate

- **ymn** (real)

Minimum Y coordinate

- **ymx** (real)

Maximum Y coordinate

- **zmn** (real)

Minimum Z coordinate

- **zmx** (real)

Maximum Z coordinate

- **heading (optional)** (string)

Title for the box

Returns

Box object

Return type

Box

Example

To create a new box in model m with label 200

```
var b = new Box(m, 200, 1.5, 2.5, 1.0, 4.5, -4.0, 3.0);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a box.

Arguments

- **Comment** (Comment)

Comment that will be attached to the box

Returns

No return value

Example

To associate comment c to the box b:

```
b.AssociateComment(c);
```

Blank()

Description

Blanks the box

Arguments

No arguments

Returns

No return value

Example

To blank box b:

```
b.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the boxs in the model.

Arguments

- **Model** (Model)

Model that all boxs will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the boxs in model m:

```
Box.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged boxs in the model.

Arguments

- **Model** (Model)

Model that all the flagged boxs will be blanked in

- **flag** (Flag)

Flag set on the boxs that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the boxs in model m flagged with f:

```
Box.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the box is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if box b is blanked:

```
if (b.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse box b:

```
b.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the box.

Arguments

- **flag** (Flag)

Flag to clear on the box

Returns

No return value

Example

To clear flag f for box b:

```
b.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the box. The target include of the copied box can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

Box object

Return type

Box

Example

To copy box b into box z:

```
var z = b.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a box

Arguments

- **Model** (Model)

Model that the box will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

Box object (or null if not made).

Return type

Box

Example

To start creating a box in model m:

```
var b = Box.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a box.

Arguments

- **Comment** (Comment)

Comment that will be detached from the box

Returns

No return value

Example

To detach comment c from the box b:

```
b.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit box b:

```
b.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for box with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The box object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for box b:

```
b.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for box b:

```
b.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for box. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for box b:

```
b.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first box in the model.

Arguments

- **Model** (Model)

Model to get first box in

Returns

Box object (or null if there are no boxs in the model).

Return type

Box

Example

To get the first box in model m:

```
var b = Box.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free box label in the model. Also see Box.LastFreeLabel(), Box.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free box label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

Box label.

Return type

Number

Example

To get the first free box label in model m:

```
var label = Box.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the boxs in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all boxs will be flagged in

- **flag** (Flag)

Flag to set on the boxs

Returns

No return value

Example

To flag all of the boxs with flag f in model m:

```
Box.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the box is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the box

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if box b has flag f set on it:

```
if (b.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each box in the model.
Note that ForEach has been designed to make looping over boxs as fast as possible and so has some limitations. Firstly, a single temporary Box object is created and on each function call it is updated with the current box data. This means that you should not try to store the Box object for later use (e.g. in an array) as it is temporary. Secondly, you cannot create new boxs inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all boxs are in

- **func** (function)

Function to call for each box

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the boxs in model m:

```
Box.ForEach(m, test);
function test(b)
{
// b is Box object
}
```

To call function test for all of the boxs in model m with optional object:

```
var data = { x:0, y:0 };
Box.ForEach(m, test, data);
function test(b, extra)
{
// b is Box object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Box objects or properties for all of the boxs in a model in PRIMER. If the optional property argument is not given then an array of Box objects is returned. If the property argument is given, that property value for each box is returned in the array instead of a Box object

Arguments

- **Model** (Model)

Model to get boxs from

- **property (optional)** (string)

Name for property to get for all boxs in the model

Returns

Array of Box objects or properties

Return type

Array

Example

To make an array of Box objects for all of the boxs in model m:

```
var a = Box.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each box in model m:

```
var a = Box.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a box.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the box b:

```
var comm_array = b.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of Box objects for all of the flagged boxs in a model in PRIMER If the optional property argument is not given then an array of Box objects is returned. If the property argument is given, then that property value for each box is returned in the array instead of a Box object

Arguments

- **Model** (Model)

Model to get boxs from

- **flag** (Flag)

Flag set on the boxs that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged boxs in the model

Returns

Array of Box objects or properties

Return type

Array

Example

To make an array of Box objects for all of the boxs in model m flagged with f:

```
var b = Box.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the boxs in model m flagged with f:

```
var a = Box.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the Box object for a box ID.

Arguments

- **Model** (Model)

Model to find the box in

- **number** (integer)

number of the box you want the Box object for

Returns

Box object (or null if box does not exist).

Return type

Box

Example

To get the Box object for box 100 in model m

```
var b = Box.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a Box property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the Box.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

box property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if Box property b.example is a parameter:

```
Options.property_parameter_names = true;
if (b.GetParameter(b.example) ) do_something...
Options.property_parameter_names = false;
```

To check if Box property b.example is a parameter by using the GetParameter method:

```
if (b.ViewParameters().GetParameter(b.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this box (*DEFINE_BOX). **Note that a carriage return is not added**. See also Box.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for box m:

```
var key = m.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the box. **Note that a carriage return is not added**. See also Box.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for box b:

```
var cards = b.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last box in the model.

Arguments

- **Model** (Model)

Model to get last box in

Returns

Box object (or null if there are no boxs in the model).

Return type

Box

Example

To get the last box in model m:

```
var b = Box.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free box label in the model. Also see Box.FirstFreeLabel(), Box.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free box label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

Box label.

Return type

Number

Example

To get the last free box label in model m:

```
var label = Box.LastFreeLabel(m);
```

Next()

Description

Returns the next box in the model.

Arguments

No arguments

Returns

Box object (or null if there are no more boxs in the model).

Return type

Box

Example

To get the box in model m after box b:

```
var b = b.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) box label in the model. Also see Box.FirstFreeLabel(), Box.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free box label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

Box label.

Return type

Number

Example

To get the next free box label in model m:

```
var label = Box.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a box.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only boxs from that model can be picked. If the argument is a Flag then only boxs that are flagged with *limit* can be selected. If omitted, or null, any boxs from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

Box object (or null if not picked)

Return type

Box

Example

To pick a box from model m giving the prompt 'Pick box from screen':

```
var b = Box.Pick('Pick box from screen', m);
```

Previous()

Description

Returns the previous box in the model.

Arguments

No arguments

Returns

Box object (or null if there are no more boxs in the model).

Return type

Box

Example

To get the box in model m before box b:

```
var b = b.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the boxs in the model.

Arguments

- **Model** (Model)

Model that all boxs will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the boxs in model m, from 1000000:

```
Box.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged boxs in the model.

Arguments

- **Model** (Model)

Model that all the flagged boxs will be renumbered in

- **flag** (Flag)

Flag set on the boxs that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the boxs in model m flagged with f, from 1000000:

```
Box.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select boxs using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting boxs

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only boxs from that model can be selected. If the argument is a Flag then only boxs that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any boxs can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of boxs selected or null if menu cancelled

Return type

Number

Example

To select boxs from model m, flagging those selected with flag f, giving the prompt 'Select boxs':

```
Box.Select(f, 'Select boxs', m);
```

To select boxs, flagging those selected with flag f but limiting selection to boxs flagged with flag l, giving the prompt 'Select boxs':

```
Box.Select(f, 'Select boxs', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the box.

Arguments

- **flag** (Flag)

Flag to set on the box

Returns

No return value

Example

To set flag f for box b:

```
b.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the box. The box will be sketched until you either call Box.Unsketch(), Box.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the box is sketched. If omitted redraw is true. If you want to sketch several boxs and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch box b:

```
b.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged boxs in the model. The boxs will be sketched until you either call Box.Unsketch(), Box.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged boxs will be sketched in

- **flag** (Flag)

Flag set on the boxs that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the boxs are sketched. If omitted redraw is true. If you want to sketch flagged boxs several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all boxs flagged with flag in model m:

```
Box.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of boxs in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing boxs should be counted. If false or omitted referenced but undefined boxs will also be included in the total.

Returns

number of boxs

Return type

Number

Example

To get the total number of boxs in model m:

```
var total = Box.Total(m);
```

Unblank()

Description

Unblanks the box

Arguments

No arguments

Returns

No return value

Example

To unblank box b:

```
b.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the boxs in the model.

Arguments

- **Model** (Model)

Model that all boxs will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the boxs in model m:

```
Box.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged boxs in the model.

Arguments

- **Model** (Model)

Model that the flagged boxs will be unblanked in

- **flag** (Flag)

Flag set on the boxs that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the boxs in model m flagged with f:

```
Box.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the boxs in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all boxs will be unset in

- **flag** (Flag)

Flag to unset on the boxs

Returns

No return value

Example

To unset the flag f on all the boxs in model m:

```
Box.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the box.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the box is unsketched. If omitted redraw is true. If you want to unsketch several boxs and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch box b:

```
b.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all boxs.

Arguments

- **Model** (Model)

Model that all boxs will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the boxs are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all boxs in model m:

```
Box.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged boxs in the model.

Arguments

- **Model** (Model)

Model that all boxs will be unsketched in

- **flag** (Flag)

Flag set on the boxs that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the boxs are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all boxs flagged with flag in model m:

```
Box.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

Box object.

Return type

Box

Example

To check if Box property b.example is a parameter by using the Box.GetParameter() method:

```
if (b.ViewParameters().GetParameter(b.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for box with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The box object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for box b:

```
b.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for box b:

```
b.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for box. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for box b:

```
b.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this box.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for box b:

```
var xrefs = b.Xrefs();
```

---

toString()

Description

Creates a string containing the box data in keyword format. Note that this contains the keyword header and the keyword cards. See also Box.Keyword() and Box.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for box b in keyword format

```
var s = b.toString();
```

ConnectionProperties class

The ConnectionProperties class gives you access to *DEFINE_CONNECTION_PROPERTIES keyword in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)

Member functions

- AddMaterialDataLine()
- AssociateComment(Comment*[Comment]*)
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetMaterialDataLine(row*[integer]*)
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- RemoveMaterialDataLine(row*[integer]*)
- SetFlag(flag*[Flag]*)
- SetMaterialDataLine(row*[integer]*, mid*[integer]*, sigy (optional)*[real]*, etan (optional)*[real]*, dg_pr (optional)*[real]*, rank (optional)*[real]*, sn (optional)*[real]*, sb (optional)*[real]*, ss (optional)*[real]*, exsn (optional)*[real]*, exsb (optional)*[real]*, exss (optional)*[real]*, lcsn (optional)*[integer]*, lcsb (optional)*[integer]*, lcss (optional)*[integer]*, gfad (optional)*[real]*, sclmrr (optional)*[real]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

ConnectionProperties properties

Name	Type	Description
add	integer	To _ADD case's parent definition
areaeq	integer	Area equation number.
con_id	integer	*DEFINE_CONNECTION_PROPERTIES id.
d_dg_pr	real	Default damage parameter.
d_dg_prf	integer	Default damage parameter(function if proprul == 2).
d_etan	real	Default tangent modulus.
d_etanf	integer	Default tangent modulus(function if proprul == 2).
d_exsb	real	Default bending stress exponent.
d_exsbf	integer	Default bending stress exponent(function if proprul == 2).
d_exsn	real	Default normal stress exponent.
d_exsnf	integer	Default normal stress exponent(function if proprul == 2).
d_exss	real	Default shear stress exponent.
d_exssf	integer	Default shear stress exponent(function if proprul == 2).
d_gfad	real	Default fading energy.
d_gfadf	integer	Default fading energy(function if proprul == 2).
d_lcsb	integer	Default LC of bending stress scale factor wrt strain rate.
d_lcsn	integer	Default LC of normal stress scale factor wrt strain rate.
d_lcss	integer	Default LC of shear stress scale factor wrt strain rate.
d_rank	real	Default rank value.
d_sb	real	Default bending strength.
d_sbf	integer	Default bending strength(function if proprul == 2).
d_sclmrr	real	Default scaling factor for torsional moment in failure function.
d_sigy	real	Default yield stress.
d_sigyf	integer	Default yield stress(function if proprul == 2).
d_sn	real	Default normal strength.
d_snf	integer	Default normal strength(function if proprul == 2).
d_ss	real	Default shear strength.
d_ssf	integer	Default shear strength(function if proprul == 2).
dg_typ	integer	Damage type.
exists (read only)	logical	true if *DEFINE_CONNECTION_PROPERTIES exists, false if referred to but not defined.
heading	string	The title of the *DEFINE_CONNECTION_PROPERTIES or the empty string if _TITLE is not set
include	integer	The Include file number that the *DEFINE_CONNECTION_PROPERTIES is in.
moarfl	integer	Modelled area flag.
model (read only)	integer	The Model number that the *DEFINE_CONNECTION_PROPERTIES is in.
proprul	integer	Property rule number.

Detailed Description

The ConnectionProperties class allows you to create, modify, edit and manipulate *DEFINE_CONNECTION_PROPERTIES. See the documentation below for more details.

Constructor

new ConnectionProperties(Model*[Model]*, con_id*[integer]*, heading (optional)*[string]*)

Description

Create a new *DEFINE_CONNECTION_PROPERTIES object.

Arguments

- **Model** (Model)

Model that *DEFINE_CONNECTION_PROPERTIES will be created in

- **con_id** (integer)

*DEFINE_CONNECTION_PROPERTIES id.

- **heading (optional)** (string)

Title for the *DEFINE_CONNECTION_PROPERTIES

Returns

ConnectionProperties object

Return type

ConnectionProperties

Example

To create a new *DEFINE_CONNECTION_PROPERTIES in model m with label 100:

```
var c = new ConnectionProperties(m, 100);
```

Details of functions

AddMaterialDataLine()

Description

Allows user to add material data line in *DEFINE_CONNECTION_PROPERTIES.

Arguments

No arguments

Returns

No return value

Example

To Add Material data line in *DEFINE_CONNECTION_PROPERTIES c:

```
c.AddMaterialDataLine();
```

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a *DEFINE_CONNECTION_PROPERTIES.

Arguments

- **Comment** (Comment)

Comment that will be attached to the *DEFINE_CONNECTION_PROPERTIES

Returns

No return value

Example

To associate comment c to the *DEFINE_CONNECTION_PROPERTIES c:

```
c.AssociateComment(c);
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse *DEFINE_CONNECTION_PROPERTIES c:

```
c.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the *DEFINE_CONNECTION_PROPERTIES.

Arguments

- **flag** (Flag)

Flag to clear on the *DEFINE_CONNECTION_PROPERTIES

Returns

No return value

Example

To clear flag f for *DEFINE_CONNECTION_PROPERTIES c:

```
c.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the *DEFINE_CONNECTION_PROPERTIES. The target include of the copied *DEFINE_CONNECTION_PROPERTIES can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

ConnectionProperties object

Return type

ConnectionProperties

Example

To copy *DEFINE_CONNECTION_PROPERTIES c into *DEFINE_CONNECTION_PROPERTIES z:

```
var z = c.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a *DEFINE_CONNECTION_PROPERTIES

Arguments

- **Model** (Model)

Model that the *DEFINE_CONNECTION_PROPERTIES will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

ConnectionProperties object (or null if not made).

Return type

ConnectionProperties

Example

To start creating a *DEFINE_CONNECTION_PROPERTIES in model m:

```
var c = ConnectionProperties.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a *DEFINE_CONNECTION_PROPERTIES.

Arguments

- **Comment** (Comment)

Comment that will be detached from the *DEFINE_CONNECTION_PROPERTIES

Returns

No return value

Example

To detach comment c from the *DEFINE_CONNECTION_PROPERTIES c:

```
c.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit *DEFINE_CONNECTION_PROPERTIES c:

```
c.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for *DEFINE_CONNECTION_PROPERTIES with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The *DEFINE_CONNECTION_PROPERTIES object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...

autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for *DEFINE_CONNECTION_PROPERTIES c:

```
c.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for *DEFINE_CONNECTION_PROPERTIES c:

```
c.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for *DEFINE_CONNECTION_PROPERTIES. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for *DEFINE_CONNECTION_PROPERTIES c:

```
c.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first *DEFINE_CONNECTION_PROPERTIES in the model.

Arguments

- **Model** (Model)

Model to get first *DEFINE_CONNECTION_PROPERTIES in

Returns

ConnectionProperties object (or null if there are no *DEFINE_CONNECTION_PROPERTIESs in the model).

Return type

ConnectionProperties

Example

To get the first *DEFINE_CONNECTION_PROPERTIES in model m:

```
var c = ConnectionProperties.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free *DEFINE_CONNECTION_PROPERTIES label in the model. Also see ConnectionProperties.LastFreeLabel(), ConnectionProperties.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free *DEFINE_CONNECTION_PROPERTIES label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

ConnectionProperties label.

Return type

Number

Example

To get the first free *DEFINE_CONNECTION_PROPERTIES label in model m:

```
var label = ConnectionProperties.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the *DEFINE_CONNECTION_PROPERTIESs in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all *DEFINE_CONNECTION_PROPERTIESs will be flagged in

- **flag** (Flag)

Flag to set on the *DEFINE_CONNECTION_PROPERTIESs

Returns

No return value

Example

To flag all of the *DEFINE_CONNECTION_PROPERTIESs with flag f in model m:

```
ConnectionProperties.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the *DEFINE_CONNECTION_PROPERTIES is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the *DEFINE_CONNECTION_PROPERTIES

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if *DEFINE_CONNECTION_PROPERTIES c has flag f set on it:

```
if (c.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each *DEFINE_CONNECTION_PROPERTIES in the model.
Note that ForEach has been designed to make looping over *DEFINE_CONNECTION_PROPERTIESs as fast as possible and so has some limitations.
Firstly, a single temporary ConnectionProperties object is created and on each function call it is updated with the current *DEFINE_CONNECTION_PROPERTIES data. This means that you should not try to store the ConnectionProperties object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new *DEFINE_CONNECTION_PROPERTIESs inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all *DEFINE_CONNECTION_PROPERTIESs are in

- **func** (function)

Function to call for each *DEFINE_CONNECTION_PROPERTIES

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the *DEFINE_CONNECTION_PROPERTIESs in model m:

```
ConnectionProperties.ForEach(m, test);
function test(c)
{
// c is ConnectionProperties object
}
```

To call function test for all of the *DEFINE_CONNECTION_PROPERTIESs in model m with optional object:

```
var data = { x:0, y:0 };
ConnectionProperties.ForEach(m, test, data);
function test(c, extra)
{
// c is ConnectionProperties object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of ConnectionProperties objects or properties for all of the *DEFINE_CONNECTION_PROPERTIESs in a model in PRIMER. If the optional property argument is not given then an array of ConnectionProperties objects is returned. If the property argument is given, that property value for each *DEFINE_CONNECTION_PROPERTIES is returned in the array instead of a ConnectionProperties object

Arguments

- **Model** (Model)

Model to get *DEFINE_CONNECTION_PROPERTIESs from

- **property (optional)** (string)

Name for property to get for all *DEFINE_CONNECTION_PROPERTIESs in the model

Returns

Array of ConnectionProperties objects or properties

Return type

Array

Example

To make an array of ConnectionProperties objects for all of the *DEFINE_CONNECTION_PROPERTIESs in model m:

```
var a = ConnectionProperties.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each *DEFINE_CONNECTION_PROPERTIES in model m:

```
var a = ConnectionProperties.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a *DEFINE_CONNECTION_PROPERTIES.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the *DEFINE_CONNECTION_PROPERTIES c:

```
var comm_array = c.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of ConnectionProperties objects for all of the flagged *DEFINE_CONNECTION_PROPERTIESs in a model in PRIMER If the optional property argument is not given then an array of ConnectionProperties objects is returned. If the property argument is given, then that property value for each *DEFINE_CONNECTION_PROPERTIES is returned in the array instead of a ConnectionProperties object

Arguments

- **Model** (Model)

Model to get *DEFINE_CONNECTION_PROPERTIESs from

- **flag** (Flag)

Flag set on the *DEFINE_CONNECTION_PROPERTIESs that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged *DEFINE_CONNECTION_PROPERTIESs in the model

Returns

Array of ConnectionProperties objects or properties

Return type

Array

Example

To make an array of ConnectionProperties objects for all of the *DEFINE_CONNECTION_PROPERTIESs in model m flagged with f:

```
var c = ConnectionProperties.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the *DEFINE_CONNECTION_PROPERTIESs in model m flagged with f:

```
var a = ConnectionProperties.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the ConnectionProperties object for a *DEFINE_CONNECTION_PROPERTIES ID.

Arguments

- **Model** (Model)

Model to find the *DEFINE_CONNECTION_PROPERTIES in

- **number** (integer)

number of the *DEFINE_CONNECTION_PROPERTIES you want the ConnectionProperties object for

Returns

ConnectionProperties object (or null if *DEFINE_CONNECTION_PROPERTIES does not exist).

Return type

ConnectionProperties

Example

To get the ConnectionProperties object for *DEFINE_CONNECTION_PROPERTIES 100 in model m

```
var c = ConnectionProperties.GetFromID(m, 100);
```

GetMaterialDataLine(row*[integer]*)

Description

Returns the material data at given row in *DEFINE_CONNECTION_PROPERTIES.

Arguments

- **row** (integer)

Material data row number, eg. for first material data, row = 0

Returns

Array of numbers containing the material id, sigy, e_tan etc. .

Return type

Number

Example

To get material data at first row, row = 0 in *DEFINE_CONNECTION_PROPERTIES c:

```
c.GetMaterialData(0);
```

GetParameter(prop*[string]*)

Description

Checks if a ConnectionProperties property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the ConnectionProperties.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

*DEFINE_CONNECTION_PROPERTIES property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if ConnectionProperties property c.example is a parameter:

```
Options.property_parameter_names = true;
if (c.GetParameter(c.example) ) do_something...
Options.property_parameter_names = false;
```

To check if ConnectionProperties property c.example is a parameter by using the GetParameter method:

```
if (c.ViewParameters().GetParameter(c.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this *DEFINE_CONNECTION_PROPERTIES **Note that a carriage return is not added**. See also ConnectionProperties.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for ConnectionProperties c:

```
var key = c.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the *DEFINE_CONNECTION_PROPERTIES. **Note that a carriage return is not added**. See also ConnectionProperties.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for connection_properties c:

```
var cards = c.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last *DEFINE_CONNECTION_PROPERTIES in the model.

Arguments

- **Model** (Model)

Model to get last *DEFINE_CONNECTION_PROPERTIES in

Returns

ConnectionProperties object (or null if there are no *DEFINE_CONNECTION_PROPERTIESs in the model).

Return type

ConnectionProperties

Example

To get the last *DEFINE_CONNECTION_PROPERTIES in model m:

```
var c = ConnectionProperties.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free *DEFINE_CONNECTION_PROPERTIES label in the model. Also see ConnectionProperties.FirstFreeLabel(), ConnectionProperties.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free *DEFINE_CONNECTION_PROPERTIES label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

ConnectionProperties label.

Return type

Number

Example

To get the last free *DEFINE_CONNECTION_PROPERTIES label in model m:

```
var label = ConnectionProperties.LastFreeLabel(m);
```

Next()

Description

Returns the next *DEFINE_CONNECTION_PROPERTIES in the model.

Arguments

No arguments

Returns

ConnectionProperties object (or null if there are no more *DEFINE_CONNECTION_PROPERTIESs in the model).

Return type

ConnectionProperties

Example

To get the *DEFINE_CONNECTION_PROPERTIES in model m after *DEFINE_CONNECTION_PROPERTIES c:

```
var c = c.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) *DEFINE_CONNECTION_PROPERTIES label in the model. Also see ConnectionProperties.FirstFreeLabel(), ConnectionProperties.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free *DEFINE_CONNECTION_PROPERTIES label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

ConnectionProperties label.

Return type

Number

Example

To get the next free *DEFINE_CONNECTION_PROPERTIES label in model m:

```
var label = ConnectionProperties.NextFreeLabel(m);
```

Previous()

Description

Returns the previous *DEFINE_CONNECTION_PROPERTIES in the model.

Arguments

No arguments

Returns

ConnectionProperties object (or null if there are no more *DEFINE_CONNECTION_PROPERTIESs in the model).

Return type

ConnectionProperties

Example

To get the *DEFINE_CONNECTION_PROPERTIES in model m before *DEFINE_CONNECTION_PROPERTIES c:

```
var c = c.Previous();
```

RemoveMaterialDataLine(row*[integer]*)

Description

Allows user to remove material data line in *DEFINE_CONNECTION_PROPERTIES.

Arguments

- **row** (integer)

Material data row number, eg. for first material data, row = 0

Returns

No return value

Example

To remove first material data line in *DEFINE_CONNECTION_PROPERTIES c:

```
c.RemoveMaterialDataLine(0);
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the *DEFINE_CONNECTION_PROPERTIESs in the model.

Arguments

- **Model** (Model)

Model that all *DEFINE_CONNECTION_PROPERTIESs will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the *DEFINE_CONNECTION_PROPERTIESs in model m, from 1000000:

```
ConnectionProperties.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged *DEFINE_CONNECTION_PROPERTIESs in the model.

Arguments

- **Model** (Model)

Model that all the flagged *DEFINE_CONNECTION_PROPERTIESs will be renumbered in

- **flag** (Flag)

Flag set on the *DEFINE_CONNECTION_PROPERTIESs that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the *DEFINE_CONNECTION_PROPERTIESs in model m flagged with f, from 1000000:

```
ConnectionProperties.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select *DEFINE_CONNECTION_PROPERTIESs using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting *DEFINE_CONNECTION_PROPERTIESs

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only *DEFINE_CONNECTION_PROPERTIESs from that model can be selected. If the argument is a Flag then only *DEFINE_CONNECTION_PROPERTIESs that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any *DEFINE_CONNECTION_PROPERTIESs can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of *DEFINE_CONNECTION_PROPERTIESs selected or null if menu cancelled

Return type

Number

Example

To select *DEFINE_CONNECTION_PROPERTIESs from model m, flagging those selected with flag f, giving the prompt 'Select *DEFINE_CONNECTION_PROPERTIESs':

```
ConnectionProperties.Select(f, 'Select *DEFINE_CONNECTION_PROPERTIESs', m);
```

To select *DEFINE_CONNECTION_PROPERTIESs, flagging those selected with flag f but limiting selection to *DEFINE_CONNECTION_PROPERTIESs flagged with flag l, giving the prompt 'Select *DEFINE_CONNECTION_PROPERTIESs':

```
ConnectionProperties.Select(f, 'Select *DEFINE_CONNECTION_PROPERTIESs', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the *DEFINE_CONNECTION_PROPERTIES.

Arguments

- **flag** (Flag)

Flag to set on the *DEFINE_CONNECTION_PROPERTIES

Returns

No return value

Example

To set flag f for *DEFINE_CONNECTION_PROPERTIES c:

```
c.SetFlag(f);
```

SetMaterialDataLine(row*[integer]*, mid*[integer]*, sigy (optional)*[real]*, etan (optional)*[real]*, dg_pr (optional)*[real]*, rank (optional)*[real]*, sn (optional)*[real]*, sb (optional)*[real]*, ss (optional)*[real]*, exsn (optional)*[real]*, exsb (optional)*[real]*, exss (optional)*[real]*, lcsn (optional)*[integer]*, lcsb (optional)*[integer]*, lcss (optional)*[integer]*, gfad (optional)*[real]*, sclmrr (optional)*[real]*)

Description

Allows user to set fields for material data line at given row in *DEFINE_CONNECTION_PROPERTIES.

Arguments

- **row** (integer)

Material data row number, eg. for first material data, row = 0

- **mid** (integer)

Material ID

- **sigy (optional)** (real)

Default yield stress

- **etan (optional)** (real)

Default tangent modulus

- **dg_pr (optional)** (real)

Default damage parameter

- **rank (optional)** (real)

Default rank value

- **sn (optional)** (real)

Default normal strength

- **sb (optional)** (real)

Default bending strength

- **ss (optional)** (real)

Default shear strength

- **exsn (optional)** (real)

Default normal stress exponent

- **exsb (optional)** (real)

Default bending stress exponent

- **exss (optional)** (real)

Default shear stress exponent

- **lcsn (optional)** (integer)

Default LC of normal stress scale factor wrt strain rate

- **lcsb (optional)** (integer)

Default LC of bending stress scale factor wrt strain rate

- **lcss (optional)** (integer)

Default LC of shear stress scale factor wrt strain rate

- **gfad (optional)** (real)

Default fading energy

- **sclmrr (optional)** (real)

Default scaling factor for torsional moment in failure function

Returns

No return value

Example

To set material data at first row (row = 0) to mat 111 in *DEFINE_CONNECTION_PROPERTIES c:

```
c.SetMaterialData(0,111);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of *DEFINE_CONNECTION_PROPERTIESs in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing *DEFINE_CONNECTION_PROPERTIESs should be counted. If false or omitted referenced but undefined *DEFINE_CONNECTION_PROPERTIESs will also be included in the total.

Returns

number of *DEFINE_CONNECTION_PROPERTIESs

Return type

Number

Example

To get the total number of *DEFINE_CONNECTION_PROPERTIESs in model m:

```
var total = ConnectionProperties.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the *DEFINE_CONNECTION_PROPERTIESs in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all *DEFINE_CONNECTION_PROPERTIESs will be unset in

- **flag** (Flag)

Flag to unset on the *DEFINE_CONNECTION_PROPERTIESs

Returns

No return value

Example

To unset the flag f on all the *DEFINE_CONNECTION_PROPERTIESs in model m:

```
ConnectionProperties.UnflagAll(m, f);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

ConnectionProperties object.

Return type

ConnectionProperties

Example

To check if ConnectionProperties property c.example is a parameter by using the ConnectionProperties.GetParameter() method:

```
if (c.ViewParameters().GetParameter(c.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for *DEFINE_CONNECTION_PROPERTIES with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The *DEFINE_CONNECTION_PROPERTIES object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for *DEFINE_CONNECTION_PROPERTIES c:

```
c.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for *DEFINE_CONNECTION_PROPERTIES c:

```
c.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for *DEFINE_CONNECTION_PROPERTIES. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for *DEFINE_CONNECTION_PROPERTIES c:

```
c.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this *DEFINE_CONNECTION_PROPERTIES.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for *DEFINE_CONNECTION_PROPERTIES c:

```
var xrefs = c.Xrefs();
```

---

toString()

Description

Creates a string containing the connection_properties data in keyword format. Note that this contains the keyword header and the keyword cards. See also ConnectionProperties.Keyword() and ConnectionProperties.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for connection_properties c in keyword format

```
var str = c.toString();
```

ConstructionStages class

The ConstructionStages class gives you access to *DEFINE_CONSTRUCTION_STAGES keyword in PRIMER.
More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

ConstructionStages properties

Name	Type	Description
ate	real	Analysis time at end of stage.
atr	real	Analysis time duration of stage.

ats	real	Analysis time at start of stage.
exists (read only)	logical	true if *DEFINE_CONSTRUCTION_STAGES exists, false if referred to but not defined.
heading	string	The title of the *DEFINE_CONSTRUCTION_STAGES or the empty string if _TITLE is not set
include	integer	The Include file number that the *DEFINE_CONSTRUCTION_STAGES is in.
istage	integer	ConstructionStages number. The label is an alternative name for this.
ivel0	integer	Flag to set velocities to zero at start of stage.
label	integer	ConstructionStages number. The istage is an alternative name for this.
model (read only)	integer	The Model number that the *DEFINE_CONSTRUCTION_STAGES is in.
rte	real	Real time at end of stage.
rts	real	Real time at start of stage.

Detailed Description

The ConstructionStages class allows you to create, modify, edit and manipulate *DEFINE_CONSTRUCTION_STAGES. See the documentation below for more details.

Constructor

new ConstructionStages(Model*[Model]*, Stage ID*[integer]*, heading (optional)*[string]*)

Description

Create a new ConstructionStages object.

Arguments

- **Model** (Model)

Model that *DEFINE_CONSTRUCTION_STAGES will be created in

- **Stage ID** (integer)

ConstructionStages id.

- **heading (optional)** (string)

Title for the *DEFINE_CONSTRUCTION_STAGES

Returns

ConstructionStages object

Return type

ConstructionStages

Example

To create a new *DEFINE_CONSTRUCTION_STAGES in model m with label 100:

```
var c = new ConstructionStages(m, 100);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a *DEFINE_CONSTRUCTION_STAGES.

Arguments

- **Comment** (Comment)

Comment that will be attached to the *DEFINE_CONSTRUCTION_STAGES

Returns

No return value

Example

To associate comment c to the *DEFINE_CONSTRUCTION_STAGES c:

```
c.AssociateComment(c);
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse *DEFINE_CONSTRUCTION_STAGES c:

```
c.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the *DEFINE_CONSTRUCTION_STAGES.

Arguments

- **flag** (Flag)

Flag to clear on the *DEFINE_CONSTRUCTION_STAGES

Returns

No return value

Example

To clear flag f for *DEFINE_CONSTRUCTION_STAGES c:

```
c.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the *DEFINE_CONSTRUCTION_STAGES. The target include of the copied *DEFINE_CONSTRUCTION_STAGES can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

ConstructionStages object

Return type

ConstructionStages

Example

To copy *DEFINE_CONSTRUCTION_STAGES c into *DEFINE_CONSTRUCTION_STAGES z:

```
var z = c.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a *DEFINE_CONSTRUCTION_STAGES

Arguments

- **Model** (Model)

Model that the *DEFINE_CONSTRUCTION_STAGES will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

ConstructionStages object (or null if not made).

Return type

ConstructionStages

Example

To start creating a *DEFINE_CONSTRUCTION_STAGES in model m:

```
var c = ConstructionStages.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a *DEFINE_CONSTRUCTION_STAGES.

Arguments

- **Comment** (Comment)

Comment that will be detached from the *DEFINE_CONSTRUCTION_STAGES

Returns

No return value

Example

To detach comment c from the *DEFINE_CONSTRUCTION_STAGES c:

```
c.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit *DEFINE_CONSTRUCTION_STAGES c:

```
c.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for *DEFINE_CONSTRUCTION_STAGES with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description

autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The *DEFINE_CONSTRUCTION_STAGES object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for *DEFINE_CONSTRUCTION_STAGES c:

```
c.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for *DEFINE_CONSTRUCTION_STAGES c:

```
c.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for *DEFINE_CONSTRUCTION_STAGES. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for *DEFINE_CONSTRUCTION_STAGES c:

```
c.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first *DEFINE_CONSTRUCTION_STAGES in the model.

Arguments

- **Model** (Model)

Model to get first *DEFINE_CONSTRUCTION_STAGES in

Returns

ConstructionStages object (or null if there are no *DEFINE_CONSTRUCTION_STAGESs in the model).

Return type

ConstructionStages

Example

To get the first *DEFINE_CONSTRUCTION_STAGES in model m:

```
var c = ConstructionStages.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free *DEFINE_CONSTRUCTION_STAGES label in the model. Also see ConstructionStages.LastFreeLabel(), ConstructionStages.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free *DEFINE_CONSTRUCTION_STAGES label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

ConstructionStages label.

Return type

Number

Example

To get the first free *DEFINE_CONSTRUCTION_STAGES label in model m:

```
var label = ConstructionStages.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the *DEFINE_CONSTRUCTION_STAGESs in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all *DEFINE_CONSTRUCTION_STAGESs will be flagged in

- **flag** (Flag)

Flag to set on the *DEFINE_CONSTRUCTION_STAGESs

Returns

No return value

Example

To flag all of the *DEFINE_CONSTRUCTION_STAGESs with flag f in model m:

```
ConstructionStages.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the *DEFINE_CONSTRUCTION_STAGES is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the *DEFINE_CONSTRUCTION_STAGES

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if *DEFINE_CONSTRUCTION_STAGES c has flag f set on it:

```
if (c.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each *DEFINE_CONSTRUCTION_STAGES in the model.
Note that ForEach has been designed to make looping over *DEFINE_CONSTRUCTION_STAGESs as fast as possible and so has some limitations.
Firstly, a single temporary ConstructionStages object is created and on each function call it is updated with the current *DEFINE_CONSTRUCTION_STAGES data. This means that you should not try to store the ConstructionStages object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new *DEFINE_CONSTRUCTION_STAGESs inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all *DEFINE_CONSTRUCTION_STAGESs are in

- **func** (function)

Function to call for each *DEFINE_CONSTRUCTION_STAGES

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the *DEFINE_CONSTRUCTION_STAGESs in model m:

```
ConstructionStages.ForEach(m, test);
function test(c)
{
// c is ConstructionStages object
}
```

To call function test for all of the *DEFINE_CONSTRUCTION_STAGESs in model m with optional object:

```
var data = { x:0, y:0 };
ConstructionStages.ForEach(m, test, data);
function test(c, extra)
{
// c is ConstructionStages object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of ConstructionStages objects or properties for all of the *DEFINE_CONSTRUCTION_STAGESs in a model in PRIMER. If the optional property argument is not given then an array of ConstructionStages objects is returned. If the property argument is given, that property value for each *DEFINE_CONSTRUCTION_STAGES is returned in the array instead of a ConstructionStages object

Arguments

- **Model** (Model)

Model to get *DEFINE_CONSTRUCTION_STAGESs from

- **property (optional)** (string)

Name for property to get for all *DEFINE_CONSTRUCTION_STAGESs in the model

Returns

Array of ConstructionStages objects or properties

Return type

Array

Example

To make an array of ConstructionStages objects for all of the *DEFINE_CONSTRUCTION_STAGESs in model m:

```
var a = ConstructionStages.GetAll(m);
```

To return an array containing the value of property ’foo’ (for example ’x’ for a node) for each *DEFINE_CONSTRUCTION_STAGES in model m:

```
var a = ConstructionStages.GetAll(m, ’foo’);
```

GetComments()

Description

Extracts the comments associated to a *DEFINE_CONSTRUCTION_STAGES.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the *DEFINE_CONSTRUCTION_STAGES c:

```
var comm_array = c.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of ConstructionStages objects for all of the flagged *DEFINE_CONSTRUCTION_STAGESs in a model in PRIMER If the optional property argument is not given then an array of ConstructionStages objects is returned. If the property argument is given, then that property value for each *DEFINE_CONSTRUCTION_STAGES is returned in the array instead of a ConstructionStages object

Arguments

- **Model** (Model)

Model to get *DEFINE_CONSTRUCTION_STAGESs from

- **flag** (Flag)

Flag set on the *DEFINE_CONSTRUCTION_STAGESs that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged *DEFINE_CONSTRUCTION_STAGESs in the model

Returns

Array of ConstructionStages objects or properties

Return type

Array

Example

To make an array of ConstructionStages objects for all of the *DEFINE_CONSTRUCTION_STAGESs in model m flagged with f:

```
var c = ConstructionStages.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the *DEFINE_CONSTRUCTION_STAGESs in model m flagged with f:

```
var a = ConstructionStages.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the ConstructionStages object for a *DEFINE_CONSTRUCTION_STAGES ID.

Arguments

- **Model** (Model)

Model to find the *DEFINE_CONSTRUCTION_STAGES in

- **number** (integer)

number of the *DEFINE_CONSTRUCTION_STAGES you want the ConstructionStages object for

Returns

ConstructionStages object (or null if *DEFINE_CONSTRUCTION_STAGES does not exist).

Return type

ConstructionStages

Example

To get the ConstructionStages object for *DEFINE_CONSTRUCTION_STAGES 100 in model m

```
var c = ConstructionStages.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a ConstructionStages property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the ConstructionStages.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

*DEFINE_CONSTRUCTION_STAGES property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if ConstructionStages property c.example is a parameter:

```
Options.property_parameter_names = true;
if (c.GetParameter(c.example) ) do_something...
Options.property_parameter_names = false;
```

To check if ConstructionStages property c.example is a parameter by using the GetParameter method:

```
if (c.ViewParameters().GetParameter(c.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this *DEFINE_CONSTRUCTION_STAGES. **Note that a carriage return is not added**. See also ConstructionStages.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for ConstructionStages c:

```
var key = c.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the *DEFINE_CONSTRUCTION_STAGES. **Note that a carriage return is not added**. See also ConstructionStages.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for construction_stages c:

```
var cards = c.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last *DEFINE_CONSTRUCTION_STAGES in the model.

Arguments

- **Model** (Model)

Model to get last *DEFINE_CONSTRUCTION_STAGES in

Returns

ConstructionStages object (or null if there are no *DEFINE_CONSTRUCTION_STAGESs in the model).

Return type

ConstructionStages

Example

To get the last *DEFINE_CONSTRUCTION_STAGES in model m:

```
var c = ConstructionStages.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free *DEFINE_CONSTRUCTION_STAGES label in the model. Also see ConstructionStages.FirstFreeLabel(), ConstructionStages.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free *DEFINE_CONSTRUCTION_STAGES label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

ConstructionStages label.

Return type

Number

Example

To get the last free *DEFINE_CONSTRUCTION_STAGES label in model m:

```
var label = ConstructionStages.LastFreeLabel(m);
```

Next()

Description

Returns the next *DEFINE_CONSTRUCTION_STAGES in the model.

Arguments

No arguments

Returns

ConstructionStages object (or null if there are no more *DEFINE_CONSTRUCTION_STAGESs in the model).

Return type

ConstructionStages

Example

To get the *DEFINE_CONSTRUCTION_STAGES in model m after *DEFINE_CONSTRUCTION_STAGES c:

```
var c = c.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) *DEFINE_CONSTRUCTION_STAGES label in the model. Also see ConstructionStages.FirstFreeLabel(), ConstructionStages.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free *DEFINE_CONSTRUCTION_STAGES label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

ConstructionStages label.

Return type

Number

Example

To get the next free *DEFINE_CONSTRUCTION_STAGES label in model m:

```
var label = ConstructionStages.NextFreeLabel(m);
```

Previous()

Description

Returns the previous *DEFINE_CONSTRUCTION_STAGES in the model.

Arguments

No arguments

Returns

ConstructionStages object (or null if there are no more *DEFINE_CONSTRUCTION_STAGESs in the model).

Return type

ConstructionStages

Example

To get the *DEFINE_CONSTRUCTION_STAGES in model m before *DEFINE_CONSTRUCTION_STAGES c:

```
var c = c.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the *DEFINE_CONSTRUCTION_STAGESs in the model.

Arguments

- **Model** (Model)

Model that all *DEFINE_CONSTRUCTION_STAGESs will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the *DEFINE_CONSTRUCTION_STAGESs in model m, from 1000000:

```
ConstructionStages.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged *DEFINE_CONSTRUCTION_STAGESs in the model.

Arguments

- **Model** (Model)

Model that all the flagged *DEFINE_CONSTRUCTION_STAGESs will be renumbered in

- **flag** (Flag)

Flag set on the *DEFINE_CONSTRUCTION_STAGESs that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the *DEFINE_CONSTRUCTION_STAGESs in model m flagged with f, from 1000000:

```
ConstructionStages.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select *DEFINE_CONSTRUCTION_STAGESs using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting *DEFINE_CONSTRUCTION_STAGESs

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only *DEFINE_CONSTRUCTION_STAGESs from that model can be selected. If the argument is a Flag then only *DEFINE_CONSTRUCTION_STAGESs that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any *DEFINE_CONSTRUCTION_STAGESs can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of *DEFINE_CONSTRUCTION_STAGESs selected or null if menu cancelled

Return type

Number

Example

To select *DEFINE_CONSTRUCTION_STAGESs from model m, flagging those selected with flag f, giving the prompt 'Select *DEFINE_CONSTRUCTION_STAGESs':

```
ConstructionStages.Select(f, 'Select *DEFINE_CONSTRUCTION_STAGESs', m);
```

To select *DEFINE_CONSTRUCTION_STAGESs, flagging those selected with flag f but limiting selection to *DEFINE_CONSTRUCTION_STAGESs flagged with flag l, giving the prompt 'Select *DEFINE_CONSTRUCTION_STAGESs':

```
ConstructionStages.Select(f, 'Select *DEFINE_CONSTRUCTION_STAGESs', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the *DEFINE_CONSTRUCTION_STAGES.

Arguments

- **flag** (Flag)

Flag to set on the *DEFINE_CONSTRUCTION_STAGES

Returns

No return value

Example

To set flag f for *DEFINE_CONSTRUCTION_STAGES c:

```
c.SetFlag(f);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of *DEFINE_CONSTRUCTION_STAGESs in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing *DEFINE_CONSTRUCTION_STAGESs should be counted. If false or omitted referenced but undefined *DEFINE_CONSTRUCTION_STAGESs will also be included in the total.

Returns

number of *DEFINE_CONSTRUCTION_STAGESs

Return type

Number

Example

To get the total number of *DEFINE_CONSTRUCTION_STAGESs in model m:

```
var total = ConstructionStages.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the *DEFINE_CONSTRUCTION_STAGESs in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all *DEFINE_CONSTRUCTION_STAGESs will be unset in

- **flag** (Flag)

Flag to unset on the *DEFINE_CONSTRUCTION_STAGESs

Returns

No return value

Example

To unset the flag f on all the *DEFINE_CONSTRUCTION_STAGESs in model m:

```
ConstructionStages.UnflagAll(m, f);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

ConstructionStages object.

Return type

ConstructionStages

Example

To check if ConstructionStages property c.example is a parameter by using the ConstructionStages.GetParameter() method:

```
if (c.ViewParameters().GetParameter(c.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for *DEFINE_CONSTRUCTION_STAGES with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The *DEFINE_CONSTRUCTION_STAGES object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for *DEFINE_CONSTRUCTION_STAGES c:

```
c.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for *DEFINE_CONSTRUCTION_STAGES c:

```
c.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for *DEFINE_CONSTRUCTION_STAGES. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for *DEFINE_CONSTRUCTION_STAGES c:

```
c.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this *DEFINE_CONSTRUCTION_STAGES.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for *DEFINE_CONSTRUCTION_STAGES c:

```
var xrefs = c.Xrefs();
```

---

toString()

Description

Creates a string containing the construction stages data in keyword format. Note that this contains the keyword header and the keyword cards. See also ConstructionStages.Keyword() and ConstructionStages.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for construction stages c in keyword format

```
var str = c.toString();
```

CoordinateSystem (Csys) class

The CoordinateSystem class gives you access to define coordinate cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]

- Xrefs()
- toString()

CoordinateSystem constants

Name	Description
CoordinateSystem.NODES	Csys is *DEFINE_COORDINATE_NODES.
CoordinateSystem.SYSTEM	Csys is *DEFINE_COORDINATE_SYSTEM.
CoordinateSystem.VECTOR	Csys is *DEFINE_COORDINATE_VECTOR.

CoordinateSystem properties

Name	Type	Description
cid	integer	CoordinateSystem number. Also see the label number.
cidl	integer	Optional local coordinate system to define the points in
dir	int	Axis defined by N1N2
exists (read only)	logical	true if coordinate system exists, false if referred to but not defined.
flag	logical	Flag for updating local system each timestep
heading	string	CoordinateSystem heading
include	integer	The Include file number that the coordinate system is in.
label	integer	CoordinateSystem number. Also see the cid property which is an alternative name for this.
lx	real	X-coordinate of point on local X-axis
ly	real	Y-coordinate of point on local X-axis
lz	real	Z-coordinate of point on local X-axis
model (read only)	integer	The Model number that the coordinate system is in.
n1	int	Node located at local origin
n2	int	Node located along local (dir) axis
n3	int	Node located in local plane determined by (dir)
nid	integer	Optional node id for rotation
option	constant	CoordinateSystem type (Can be CoordinateSystem.NODES, CoordinateSystem.SYSTEM or CoordinateSystem.VECTOR).
ox	real	X-coordinate of origin
oy	real	Y-coordinate of origin
oz	real	Z-coordinate of origin
px	real	X-coordinate of point in local X-Y plane
py	real	Y-coordinate of point in local X-Y plane
pz	real	Z-coordinate of point in local X-Y plane
vx	real	X-coordinate of local X-Y vector
vy	real	Y-coordinate of local X-Y vector
vz	real	Z-coordinate of local X-Y vector
xx	real	X-coordinate on local X-axis

xy	real	Y-coordinate on local X-axis
xz	real	Z-coordinate on local X-axis

Detailed Description

The CoordinateSystem class allows you to create, modify, edit and manipulate csys cards. See the documentation below for more details.

For convenience "Csys" can also be used as the class name instead of "CoordinateSystem".

Constructor

new CoordinateSystem(Model*[Model]*, details*[object]*)

Description

Create a new CoordinateSystem object for *DEFINE_COORDINATE_NODES.

Arguments

- **Model** (Model)

Model that csys will be created in

- **details** (object)

Details for creating the CoordinateSystem

Object has the following properties:

Name	Type	Description
cid	integer	Label of CoordinateSystem
cl (optional)	Array of numbers	Array of coordinates of point on local X-axis [lx, ly, lz] (for option CoordinateSystem.SYSTEM)
co (optional)	Array of numbers	Array of coordinates of origin [ox, oy, oz] (for option CoordinateSystem.SYSTEM)
cp (optional)	Array of numbers	Array of coordinates of point in local X-Y plane [px, py, pz] (for option CoordinateSystem.SYSTEM)
cv (optional)	Array of numbers	Array of coordinates of local X-Y vector [vx, vy, vz] (for option CoordinateSystem.VECTOR)
cx (optional)	Array of numbers	Array of coordinates on local X-axis [xx, xy, xz] (for option CoordinateSystem.VECTOR)
dir (optional)	integer	Axis defined by N1N2 (for option CoordinateSystem.NODES)
flag (optional)	boolean	Flag for local system update for each time step (for option CoordinateSystem.NODES)
heading (optional)	string	Title for the coordinate system
nid (optional)	integer	Optional Node ID for rotation (for option CoordinateSystem.VECTOR)
nodes (optional)	Array of numbers	Array of Node IDs [n1, n2, n3] for the coordinate system (for option CoordinateSystem.NODES)
option	constant	CoordinateSystem type (can be CoordinateSystem.NODES, CoordinateSystem.SYSTEM or CoordinateSystem.VECTOR)

Returns

CoordinateSystem object

Return type

CoordinateSystem

Example

To create a new Csys of type Nodes in model m with label 200 and title "Test csys 1" defined by nodes 1, 2, 3 with where N1N2 defines local Y-axis; local system update flag is off

```
var c = new CoordinateSystem(m, {option: CoordinateSystem.NODES, cid: 200,
nodes: [1, 2, 3], flag: 0, dir: 2, heading: "Test csys nodes"});
```

To create a new Csys of type System in model m with label 300 and title "Test csys 2" with origin at (10, 10, 0), point on local X-axis at (20, 20, 0) and point in X-Y plane at (10, 20, 0)

```
var c = new CoordinateSystem(m, {option: CoordinateSystem.SYSTEM, cid: 300, co:
[10, 10, 0], cl: [20, 20, 0], cp: [10, 20, 0], heading: "Test csys system"});
```

To create a new Csys of type Vector in model m with label 400 with point on local X-axis at (50, 50, 0) and local XY vector being (-10, -20, 0) that can rotate with node 10003

```
var c = new CoordinateSystem(m, {option: CoordinateSystem.VECTOR, cid: 400, cx:
[50, 50, 0], cv: [10, -20, 0], nid: 10003, heading: "Test csys vector"});
```

new CoordinateSystem(Model*[Model]*, option*[constant]*, cid*[integer]*, n1*[integer]*, n2*[integer]*, n3*[integer]*, flag*[boolean]*, dir*[integer]*, heading (optional)*[string]*) [deprecated]

This function is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Create a new CoordinateSystem object for *DEFINE_COORDINATE_NODES.

Arguments

- **Model** (Model)

Model that csys will be created in

- **option** (constant)

Must be CoordinateSystem.NODES

- **cid** (integer)

CoordinateSystem number

- **n1** (integer)

Node located at origin

- **n2** (integer)

Node located along (DIR) axis

- **n3** (integer)

Node located in plane defined by (DIR)

- **flag** (boolean)

Flag for local system update each time step

- **dir** (integer)

Axis defined by N1N2

- **heading (optional)** (string)

Title for the csys

Returns

CoordinateSystem object

Return type

CoordinateSystem

Example

To create a new Csys of type Nodes in model m with label 200 and title "Test csys 1" defined by nodes 1, 2, 3 with where N1N2 defines local Y-axis; local system update flag is off

```
var c = new CoordinateSystem(m, CoordinateSystem.NODES, 200, 1, 2, 3, 0, 2,
"Test csys");
```

new CoordinateSystem(Model*[Model]*, option*[constant]*, cid*[integer]*, ox*[real]*, oy*[real]*, oz*[real]*, lx*[real]*, ly*[real]*, lz*[real]*, px*[real]*, py*[real]*, pz*[real]*, heading (optional)*[string]*) [deprecated]

This function is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Create a new CoordinateSystem object for *DEFINE_COORDINATE_SYSTEM.

Arguments

- **Model** (Model)

Model that csys will be created in

- **option** (constant)

Must be CoordinateSystem.SYSTEM

- **cid** (integer)

CoordinateSystem number

- **ox** (real)

X-coordinate of origin

- **oy** (real)

Y-coordinate of origin

- **oz** (real)

Z-coordinate of origin

- **lx** (real)

X-coordinate of point on local X-axis

- **ly** (real)

Y-coordinate of point on local X-axis

- **lz** (real)

Z-coordinate of point on local X-axis

- **px** (real)

X-coordinate of point in local X-Y plane

- **py** (real)

Y-coordinate of point in local X-Y plane

- **pz** (real)

Z-coordinate of point in local X-Y plane

- **heading (optional)** (string)

Title for the csys

Returns

No return value

Example

To create a new Csys of type System in model m with label 300 and title "Test csys 2" with origin at (10, 10, 0), point on local X-axis at (20, 20, 0) and point on X-y at (10, 20, 0)

```
var c = new CoordinateSystem(m, CoordinateSystem.SYSTEM, 300, 10, 10, 0, 20, 20, 0, 10, 20, 0, "Test csys");
```

new CoordinateSystem(Model*[Model]*, option*[constant]*, cid*[integer]*, xx*[real]*, xy*[real]*, xz*[real]*, vx*[real]*, vy*[real]*, vz*[real]*, nid (optional)*[integer]*, heading (optional)*[string]*) [deprecated]

This function is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Create a new CoordinateSystem object for *DEFINE_COORDINATE_VECTOR.

Arguments

- **Model** (Model)

Model that csys will be created in

- **option** (constant)

Must be CoordinateSystem.VECTOR

- **cid** (integer)

CoordinateSystem number

- **xx** (real)

X-coordinate on local X-axis

- **xy** (real)

Y-coordinate on local X-axis

- **xz** (real)

Z-coordinate on local X-axis

- **vx** (real)

X-coordinate of local X-Y vector

- **vy** (real)

Y-coordinate of local X-Y vector

- **vz** (real)

Z-coordinate of local X-Z vector

- **nid (optional)** (integer)

Optional node id for rotation

- **heading (optional)** (string)

Title for the csys

Returns

No return value

Example

To create a new Csys of type Vector in model m with label 400 with point on local X-axis at (50, 50, 0) and local XY being (-10, -20, 0) that can rotate with node 10003

```
var c = new CoordinateSystem(m, CoordinateSystem.VECTOR, 400, 50, 50, 0, 10, -20
, 0, 10003);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a coordinate system.

Arguments

- **Comment** (Comment)

Comment that will be attached to the coordinate system

Returns

No return value

Example

To associate comment c to the coordinate system c:

```
c.AssociateComment(c);
```

Blank()

Description

Blanks the coordinate system

Arguments

No arguments

Returns

No return value

Example

To blank coordinate system c:

```
c.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the coordinate systems in the model.

Arguments

- **Model** (Model)

Model that all coordinate systems will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after

the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the coordinate systems in model m:

```
CoordinateSystem.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged coordinate systems in the model.

Arguments

- **Model** (Model)

Model that all the flagged coordinate systems will be blanked in

- **flag** (Flag)

Flag set on the coordinate systems that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the coordinate systems in model m flagged with f:

```
CoordinateSystem.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the coordinate system is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if coordinate system c is blanked:

```
if (c.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse coordinate system c:

```
c.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the coordinate system.

Arguments

- **flag** (Flag)

Flag to clear on the coordinate system

Returns

No return value

Example

To clear flag f for coordinate system c:

```
c.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the coordinate system. The target include of the copied coordinate system can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

CoordinateSystem object

Return type

CoordinateSystem

Example

To copy coordinate system c into coordinate system z:

```
var z = c.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a coordinate system

Arguments

- **Model** (Model)

Model that the coordinate system will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

CoordinateSystem object (or null if not made).

Return type

CoordinateSystem

Example

To start creating a coordinate system in model m:

```
var c = CoordinateSystem.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a coordinate system.

Arguments

- **Comment** (Comment)

Comment that will be detached from the coordinate system

Returns

No return value

Example

To detach comment c from the coordinate system c:

```
c.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit coordinate system c:

```
c.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for coordinate system with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The coordinate system object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for coordinate system c:

```
c.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for coordinate system c:

```
c.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for coordinate system. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for coordinate system c:

```
c.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first coordinate system in the model.

Arguments

- **Model** (Model)

Model to get first coordinate system in

Returns

CoordinateSystem object (or null if there are no coordinate systems in the model).

Return type

CoordinateSystem

Example

To get the first coordinate system in model m:

```
var c = CoordinateSystem.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free coordinate system label in the model. Also see CoordinateSystem.LastFreeLabel(), CoordinateSystem.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free coordinate system label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

CoordinateSystem label.

Return type

Number

Example

To get the first free coordinate system label in model m:

```
var label = CoordinateSystem.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the coordinate systems in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all coordinate systems will be flagged in

- **flag** (Flag)

Flag to set on the coordinate systems

Returns

No return value

Example

To flag all of the coordinate systems with flag f in model m:

```
CoordinateSystem.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the coordinate system is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the coordinate system

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if coordinate system c has flag f set on it:

```
if (c.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each coordinate system in the model.
Note that ForEach has been designed to make looping over coordinate systems as fast as possible and so has some limitations.
Firstly, a single temporary CoordinateSystem object is created and on each function call it is updated with the current coordinate system data. This means that you should not try to store the CoordinateSystem object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new coordinate systems inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all coordinate systems are in

- **func** (function)

Function to call for each coordinate system

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the coordinate systems in model m:

```
CoordinateSystem.ForEach(m, test);
function test(c)
{
// c is CoordinateSystem object
}
```

To call function test for all of the coordinate systems in model m with optional object:

```
var data = { x:0, y:0 };
CoordinateSystem.ForEach(m, test, data);
function test(c, extra)
{
// c is CoordinateSystem object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of CoordinateSystem objects or properties for all of the coordinate systems in a model in PRIMER. If the optional property argument is not given then an array of CoordinateSystem objects is returned. If the property argument is given, that property value for each coordinate system is returned in the array instead of a CoordinateSystem object

Arguments

- **Model** (Model)

Model to get coordinate systems from

- **property (optional)** (string)

Name for property to get for all coordinate systems in the model

Returns

Array of CoordinateSystem objects or properties

Return type

Array

Example

To make an array of CoordinateSystem objects for all of the coordinate systems in model m:

```
var a = CoordinateSystem.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each coordinate system in model m:

```
var a = CoordinateSystem.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a coordinate system.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the coordinate system c:

```
var comm_array = c.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of CoordinateSystem objects for all of the flagged coordinate systems in a model in PRIMER If the optional property argument is not given then an array of CoordinateSystem objects is returned. If the property argument is given, then that property value for each coordinate system is returned in the array instead of a CoordinateSystem object

Arguments

- **Model** (Model)

Model to get coordinate systems from

- **flag** (Flag)

Flag set on the coordinate systems that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged coordinate systems in the model

Returns

Array of CoordinateSystem objects or properties

Return type

Array

Example

To make an array of CoordinateSystem objects for all of the coordinate systems in model m flagged with f:

```
var c = CoordinateSystem.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the coordinate systems in model m flagged with f:

```
var a = CoordinateSystem.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the CoordinateSystem object for a coordinate system ID.

Arguments

- **Model** (Model)

Model to find the coordinate system in

- **number** (integer)

number of the coordinate system you want the CoordinateSystem object for

Returns

CoordinateSystem object (or null if coordinate system does not exist).

Return type

CoordinateSystem

Example

To get the CoordinateSystem object for coordinate system 100 in model m

```
var c = CoordinateSystem.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a CoordinateSystem property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the CoordinateSystem.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

coordinate system property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if CoordinateSystem property c.example is a parameter:

```
Options.property_parameter_names = true;
if (c.GetParameter(c.example) ) do_something...
Options.property_parameter_names = false;
```

To check if CoordinateSystem property c.example is a parameter by using the GetParameter method:

```
if (c.ViewParameters().GetParameter(c.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this csys (*DEFINE_COORDINATE). **Note that a carriage return is not added**. See also CoordinateSystem.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for csys m:

```
var key = m.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the csys. **Note that a carriage return is not added**. See also CoordinateSystem.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for csys c:

```
var cards = v.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last coordinate system in the model.

Arguments

- **Model** (Model)

Model to get last coordinate system in

Returns

CoordinateSystem object (or null if there are no coordinate systems in the model).

Return type

CoordinateSystem

Example

To get the last coordinate system in model m:

```
var c = CoordinateSystem.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free coordinate system label in the model. Also see CoordinateSystem.FirstFreeLabel(), CoordinateSystem.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free coordinate system label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

CoordinateSystem label.

Return type

Number

Example

To get the last free coordinate system label in model m:

```
var label = CoordinateSystem.LastFreeLabel(m);
```

Next()

Description

Returns the next coordinate system in the model.

Arguments

No arguments

Returns

CoordinateSystem object (or null if there are no more coordinate systems in the model).

Return type

CoordinateSystem

Example

To get the coordinate system in model m after coordinate system c:

```
var c = c.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) coordinate system label in the model. Also see CoordinateSystem.FirstFreeLabel(), CoordinateSystem.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free coordinate system label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

CoordinateSystem label.

Return type

Number

Example

To get the next free coordinate system label in model m:

```
var label = CoordinateSystem.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a coordinate system.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only coordinate systems from that model can be picked. If the argument is a Flag then only coordinate systems that are flagged with *limit* can be selected. If omitted, or null, any coordinate systems from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

CoordinateSystem object (or null if not picked)

Return type

CoordinateSystem

Example

To pick a coordinate system from model m giving the prompt 'Pick coordinate system from screen':

```
var c = CoordinateSystem.Pick('Pick coordinate system from screen', m);
```

Previous()

Description

Returns the previous coordinate system in the model.

Arguments

No arguments

Returns

CoordinateSystem object (or null if there are no more coordinate systems in the model).

Return type

CoordinateSystem

Example

To get the coordinate system in model m before coordinate system c:

```
var c = c.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the coordinate systems in the model.

Arguments

- **Model** (Model)

Model that all coordinate systems will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the coordinate systems in model m, from 1000000:

```
CoordinateSystem.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged coordinate systems in the model.

Arguments

- **Model** (Model)

Model that all the flagged coordinate systems will be renumbered in

- **flag** (Flag)

Flag set on the coordinate systems that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the coordinate systems in model m flagged with f, from 1000000:

```
CoordinateSystem.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select coordinate systems using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting coordinate systems

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only coordinate systems from that model can be selected. If the argument is a Flag then only coordinate systems that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any coordinate systems can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of coordinate systems selected or null if menu cancelled

Return type

Number

Example

To select coordinate systems from model m, flagging those selected with flag f, giving the prompt 'Select coordinate systems':

```
CoordinateSystem.Select(f, 'Select coordinate systems', m);
```

To select coordinate systems, flagging those selected with flag f but limiting selection to coordinate systems flagged with flag l, giving the prompt 'Select coordinate systems':

```
CoordinateSystem.Select(f, 'Select coordinate systems', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the coordinate system.

Arguments

- **flag** (Flag)

Flag to set on the coordinate system

Returns

No return value

Example

To set flag f for coordinate system c:

```
c.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the coordinate system. The coordinate system will be sketched until you either call CoordinateSystem.Unsketch(), CoordinateSystem.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the coordinate system is sketched. If omitted redraw is true. If you want to sketch several coordinate systems and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch coordinate system c:

```
c.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged coordinate systems in the model. The coordinate systems will be sketched until you either call CoordinateSystem.Unsketch(), CoordinateSystem.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged coordinate systems will be sketched in

- **flag** (Flag)

Flag set on the coordinate systems that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the coordinate systems are sketched. If omitted redraw is true. If you want to sketch flagged coordinate systems several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all coordinate systems flagged with flag in model m:

```
CoordinateSystem.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of coordinate systems in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing coordinate systems should be counted. If false or omitted referenced but undefined coordinate systems will also be included in the total.

Returns

number of coordinate systems

Return type

Number

Example

To get the total number of coordinate systems in model m:

```
var total = CoordinateSystem.Total(m);
```

Unblank()

Description

Unblanks the coordinate system

Arguments

No arguments

Returns

No return value

Example

To unblank coordinate system c:

```
c.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the coordinate systems in the model.

Arguments

- **Model** (Model)

Model that all coordinate systems will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the coordinate systems in model m:

```
CoordinateSystem.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged coordinate systems in the model.

Arguments

- **Model** (Model)

Model that the flagged coordinate systems will be unblanked in

- **flag** (Flag)

Flag set on the coordinate systems that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the coordinate systems in model m flagged with f:

```
CoordinateSystem.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the coordinate systems in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all coordinate systems will be unset in

- **flag** (Flag)

Flag to unset on the coordinate systems

Returns

No return value

Example

To unset the flag f on all the coordinate systems in model m:

```
CoordinateSystem.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the coordinate system.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the coordinate system is unsketched. If omitted redraw is true. If you want to unsketch several coordinate systems and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch coordinate system c:

```
c.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all coordinate systems.

Arguments

- **Model** (Model)

Model that all coordinate systems will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the coordinate systems are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all coordinate systems in model m:

```
CoordinateSystem.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged coordinate systems in the model.

Arguments

- **Model** (Model)

Model that all coordinate systems will be unsketched in

- **flag** (Flag)

Flag set on the coordinate systems that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the coordinate systems are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all coordinate systems flagged with flag in model m:

```
CoordinateSystem.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

CoordinateSystem object.

Return type

CoordinateSystem

Example

To check if CoordinateSystem property c.example is a parameter by using the CoordinateSystem.GetParameter() method:

```
if (c.ViewParameters().GetParameter(c.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for coordinate system with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The coordinate system object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for coordinate system c:

```
c.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for coordinate system c:

```
c.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for coordinate system. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for coordinate system c:

```
c.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this coordinate system.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for coordinate system c:

```
var xrefs = c.Xrefs();
```

---

toString()

Description

Creates a string containing the csys data in keyword format. Note that this contains the keyword header and the keyword cards. See also CoordinateSystem.Keyword() and CoordinateSystem.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for csys c in keyword format

```
var s = v.toString();
```

Curve class

The Curve class gives you access to load curve cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Create(Model*[Model]*, modal (optional)*[boolean]*)
- CreateTable(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)

Member functions

- AddPoint(xvalue*[real]*, yvalue*[real]*)
- AddTableEntry(value*[real]*, load curve*[integer]*)
- AssociateComment(Comment*[Comment]*)
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- GetPoint(row*[integer]*)
- GetTableEntry(row*[integer]*)
- InsertPoint(ipt*[integer]*, xvalue*[real]*, yvalue*[real]*, position*[integer]*)
- InsertTableEntry(ipt*[integer]*, value*[real]*, lcid*[integer]*, position*[integer]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- RemovePoint(row*[integer]*)
- RemoveTableEntry(ipt*[integer]*)
- SetFlag(flag*[Flag]*)
- SetPoint(ipt*[integer]*, xvalue*[real]*, yvalue*[real]*)
- SetTableEntry(ipt*[integer]*, value*[real]*, load curve*[integer]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

Curve constants

Name	Description
Curve.AFTER	Insertion of curve data option.
Curve.BEFORE	Insertion of curve data option.
Curve.CURVE	Load curve type *DEFINE_CURVE
Curve.CURVE_FUNCTION	Load curve type *DEFINE_CURVE_FUNCTION
Curve.CURVE_SMOOTH	Load curve type *DEFINE_CURVE_SMOOTH
Curve.FUNCTION	Load curve type *DEFINE_FUNCTION
Curve.TABLE	Load curve type *DEFINE_TABLE

Curve properties

Name	Type	Description
dattyp	integer	Data type
dist	real	Total distance tool will travel
exists (read only)	logical	true if curve exists, false if referred to but not defined.
function	string	Function expression for Curve.CURVE_FUNCTION
heading	string	Curve heading
include	integer	The Include file number that the curve is in.
label	integer	Curve number. Also see the lcid property which is an alternative name for this.
lcid	integer	Curve number. Also see the label property which is an alternative name for this.
lcint	integer	Number of discretization points for the curve
model (read only)	integer	The Model number that the curve is in.
ncurves	integer	Number of points in curve or number of curves in table. The npoints property is an alternative name for this. (read only for tables)
npoints	integer	Number of points in curve or number of curves in table. The ncurves property is an alternative name for this. (read only for tables)
offa	real	Offset for abscissa values
offo	real	Offset for ordinate values
sfa	real	Scale factor on abscissa value
sfo	real	Scale factor on ordinate value
sidr	integer	Stress initialisation by dynamic relaxation
tend	real	Time curve returns to zero
trise	real	Rise time
tstart	real	Time curve starts to rise
type	constant	Load curve type (Can be Curve.CURVE, Curve.CURVE_FUNCTION, Curve.SMOOTH, Curve.FUNCTION or Curve.TABLE).
version	string	Version for discretization. Can be blank, "3858" or "5434a"
vmax	real	Maximum velocity

Detailed Description

The Curve class allows you to create, modify, edit and manipulate curve cards. See the documentation below for more details.

Constructor

new Curve(Model*[Model]*, options *[object]*)

Description

Create a new Curve object.

Arguments

- **Model** (Model)

Model that curve will be created in

- **options** (object)

Options for creating the curve

Object has the following properties:

Name	Type	Description
dattyp (optional)	integer	Data type
dist (optional)	real	Total distance tool will travel (Type of load curve must be Curve.CURVE_SMOOTH)
function (optional)	string	Function expression (load curve type has to be Curve.FUNCTION or Curve.CURVE_FUNCTION)
heading (optional)	string	Title for the curve
lcid	integer	Curve number
lcint (optional)	integer	Data type
offa (optional)	real	Offset on abscissa value
offo (optional)	real	Offset on ordinate value
sfa (optional)	real	Scale factor on abscissa value
sfo (optional)	real	Scale factor on ordinate value
sidr (optional)	integer	Stress initialisation by dynamic relaxation
tend (optional)	real	Time curve returns to zero (Type of load curve must be Curve.CURVE_SMOOTH)
trise (optional)	real	Rise time (Type of load curve must be Curve.CURVE_SMOOTH)
tstart (optional)	real	Time curve starts to rise (Type of load curve must be Curve.CURVE_SMOOTH)

type	constant	Type of load curve. Can be Curve.CURVE, Curve.TABLE, Note this does not have to be defined. In previous versions of PRIMER you could only construct a basic load curve type, therefore the type argument was not used. PRIMER is still backwards compatible with this method of load curve creation.
vmax (optional)	real	Maximum velocity (Type of load curve must be Curve.CURVE_SMOOTH)

Returns

Curve object

Return type

Curve

Example

To create a new curve in model m with label 200

```
var l = new Curve(Curve.CURVE, m, 200);
```

To create a new curve function in model m with label 200 and function ’0.5*lc9*vm(22)**3’ (example from keyword manual)

```
var l = new Curve(Curve.CURVE_FUNCTION, m, 200, 0, "0.5*lc9*vm(22)**3");
```

To create a new function in model m with label 200 and function ’x(t)=1000*sin(100*t)’ and title ’x-velo’ (example from keyword manual)

```
var l = new Curve(Curve.FUNCTION, m, 200, "x(t)=1000*sin(100*t)", "x-velo");
```

To create a new curve function in model m with label 200’ (example from keyword manual)

```
var l = new Curve(Curve.CURVE_SMOOTH, m, 200);
```

new Curve(Load curve type*[constant]*, Model*[Model]*, lcid*[integer]*, sidr (optional)*[integer]*, sfa (optional)*[real]*, sfo (optional)*[real]*, offa (optional)*[real]*, offo (optional)*[real]*, dattyp (optional)*[integer]*, heading (optional)*[string]*, lcint (optional)*[integer]*) [deprecated]

This function is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Create a new Curve object.

Arguments

- **Load curve type** (constant)

Type of load curve. Can be Curve.CURVE, Curve.TABLE, Note this does not have to be defined. In previous versions of PRIMER you could only construct a basic load curve type, therefore the type argument was not used. PRIMER is still backwards compatible with this method of load curve creation.

- **Model** (Model)

Model that curve will be created in

- **lcid** (integer)

Curve number

- **sidr (optional)** (integer)

Stress initialisation by dynamic relaxation

- **sfa (optional)** (real)

Scale factor on abscissa value

- **sfo (optional)** (real)

Scale factor on ordinate value

- **offa (optional)** (real)

Offset on abscissa value

- **offo (optional)** (real)

Offset on ordinate value

- **dattyp (optional)** (integer)

Data type

- **heading (optional)** (string)

Title for the curve

- **lcint (optional)** (integer)

Number of discretization points for the curve

Returns

Curve object

Return type

Curve

Example

To create a new curve in model m with label 200

```
var l = new Curve(Curve.CURVE, m, 200);
```

new Curve(Load curve type*[constant]*, Model*[Model]*, lcid*[integer]*, sidr (optional)*[integer]*, function (optional)*[string]*, heading (optional)*[string]*) [deprecated]

This function is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Create a new Curve *DEFINE_CURVE_FUNCTION object.

Arguments

- **Load curve type** (constant)

Type of load curve. Must be Curve.CURVE_FUNCTION.

- **Model** (Model)

Model that curve will be created in

- **lcid** (integer)

Curve number

- **sidr (optional)** (integer)

Stress initialisation by dynamic relaxation

- **function (optional)** (string)

Function expression

- **heading (optional)** (string)

Title for the curve

Returns

Curve object

Return type

Curve

Example

To create a new curve function in model m with label 200 and function ’0.5*lc9*vm(22)**3’ (example from keyword manual)

```
var l = new Curve(Curve.CURVE_FUNCTION, m, 200, 0, "0.5*lc9*vm(22)**3");
```

new Curve(Load curve type*[constant]*, Model*[Model]*, lcid*[integer]*, function (optional)*[string]*, heading (optional)*[string]*) [deprecated]

This function is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Create a new Curve *DEFINE_FUNCTION object.

Arguments

- **Load curve type** (constant)

Type of load curve. Must be Curve.FUNCTION.

- **Model** (Model)

Model that curve will be created in

- **lcid** (integer)

Curve number

- **function (optional)** (string)

Function expression

- **heading (optional)** (string)

Title for the curve

Returns

Curve object

Return type

Curve

Example

To create a new function in model m with label 200 and function ’x(t)=1000*sin(100*t)’ and title ’x-velo’ (example from keyword manual)

```
var l = new Curve(Curve.FUNCTION, m, 200, "x(t)=1000*sin(100*t)", "x-velo");
```

new Curve(Load curve type*[constant]*, Model*[Model]*, lcid*[integer]*, sidr (optional)*[integer]*, dist (optional)*[real]*, tstart (optional)*[real]*, tend (optional)*[real]*, trise (optional)*[real]*, vmax (optional)*[real]*, heading (optional)*[string]*) [deprecated]

This function is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Create a new Curve *DEFINE_CURVE_SMOOTH object.

Arguments

- **Load curve type** (constant)

Type of load curve. Must be Curve.CURVE_SMOOTH.

- **Model** (Model)

Model that curve will be created in

- **lcid** (integer)

Curve number

- **sidr (optional)** (integer)

Stress initialisation by dynamic relaxation

- **dist (optional)** (real)

Total distance tool will travel

- **tstart (optional)** (real)

Time curve starts to rise

- **tend (optional)** (real)

Time curve returns to zero

- **trise (optional)** (real)

Rise time

- **vmax (optional)** (real)

Maximum velocity

- **heading (optional)** (string)

Title for the curve

Returns

Curve object

Return type

Curve

Example

To create a new curve function in model m with label 200’ (example from keyword manual)

```
var l = new Curve(Curve.CURVE_SMOOTH, m, 200);
```

Details of functions

AddPoint(xvalue*[real]*, yvalue*[real]*)

Description

Adds a point to a load curve.

Arguments

- **xvalue** (real)

The x value of the point.

- **yvalue** (real)

The y value of the point.

Returns

No return value.

Example

To add a point with values of x=3 and y=5 to curve l:

```
l.AddPoint(3, 5);
```

AddTableEntry(value*[real]*, load curve*[integer]*)

Description

Adds an entry line to a table.

Arguments

- **value** (real)

The value for for this entry in the table.

- **load curve** (integer)

The load curve corresponding to the defined value.

Returns

No return value.

Example

To add an entry with a value of 3 for load curve 1000:

```
l.AddTableEntry(3, 1000);
```

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a curve.

Arguments

- **Comment** (Comment)

Comment that will be attached to the curve

Returns

No return value

Example

To associate comment c to the curve c:

```
c.AssociateComment(c);
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse curve c:

```
c.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the curve.

Arguments

- **flag** (Flag)

Flag to clear on the curve

Returns

No return value

Example

To clear flag f for curve c:

```
c.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the curve. The target include of the copied curve can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

Curve object

Return type

Curve

Example

To copy curve c into curve z:

```
var z = c.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a curve.

Arguments

- **Model** (Model)

Model that the curve will be created in

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

Curve object (or null if not made)

Return type

Curve

Example

To start creating a curve in model m:

```
var l = Curve.Create(m);
```

CreateTable(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a table.

Arguments

- **Model** (Model)

Model that the curve will be created in

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

Curve object (or null if not made)

Return type

Curve

Example

To start creating a table in model m:

```
var l = Curve.CreateTable(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a curve.

Arguments

- **Comment** (Comment)

Comment that will be detached from the curve

Returns

No return value

Example

To detach comment c from the curve c:

```
c.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit curve c:

```
c.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for curve with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The curve object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for curve c:

```
c.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for curve c:

```
c.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for curve. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for curve c:

```
c.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first curve in the model.

Arguments

- **Model** (Model)

Model to get first curve in

Returns

Curve object (or null if there are no curves in the model).

Return type

Curve

Example

To get the first curve in model m:

```
var c = Curve.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free curve label in the model. Also see Curve.LastFreeLabel(), Curve.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free curve label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

Curve label.

Return type

Number

Example

To get the first free curve label in model m:

```
var label = Curve.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the curves in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all curves will be flagged in

- **flag** (Flag)

Flag to set on the curves

Returns

No return value

Example

To flag all of the curves with flag f in model m:

```
Curve.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the curve is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the curve

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if curve c has flag f set on it:

```
if (c.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each curve in the model.
Note that ForEach has been designed to make looping over curves as fast as possible and so has some limitations. Firstly, a single temporary Curve object is created and on each function call it is updated with the current curve data. This means that you should not try to store the Curve object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new curves inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all curves are in

- **func** (function)

Function to call for each curve

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the curves in model m:

```
Curve.ForEach(m, test);
function test(c)
{
// c is Curve object
}
```

To call function test for all of the curves in model m with optional object:

```
var data = { x:0, y:0 };
Curve.ForEach(m, test, data);
function test(c, extra)
{
// c is Curve object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Curve objects or properties for all of the curves in a model in PRIMER. If the optional property argument is not given then an array of Curve objects is returned. If the property argument is given, that property value for each curve is returned in the array instead of a Curve object

Arguments

- **Model** (Model)

Model to get curves from

- **property (optional)** (string)

Name for property to get for all curves in the model

Returns

Array of Curve objects or properties

Return type

Array

Example

To make an array of Curve objects for all of the curves in model m:

```
var a = Curve.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each curve in model m:

```
var a = Curve.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a curve.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the curve c:

```
var comm_array = c.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of Curve objects for all of the flagged curves in a model in PRIMER If the optional property argument is not given then an array of Curve objects is returned. If the property argument is given, then that property value for each curve is returned in the array instead of a Curve object

Arguments

- **Model** (Model)

Model to get curves from

- **flag** (Flag)

Flag set on the curves that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged curves in the model

Returns

Array of Curve objects or properties

Return type

Array

Example

To make an array of Curve objects for all of the curves in model m flagged with f:

```
var c = Curve.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the curves in model m flagged with f:

```
var a = Curve.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the Curve object for a curve ID.

Arguments

- **Model** (Model)

Model to find the curve in

- **number** (integer)

number of the curve you want the Curve object for

Returns

Curve object (or null if curve does not exist).

Return type

Curve

Example

To get the Curve object for curve 100 in model m

```
var c = Curve.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a Curve property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the Curve.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

curve property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if Curve property c.example is a parameter:

```
Options.property_parameter_names = true;
if (c.GetParameter(c.example) ) do_something...
Options.property_parameter_names = false;
```

To check if Curve property c.example is a parameter by using the GetParameter method:

```
if (c.ViewParameters().GetParameter(c.example) ) do_something...
```

GetPoint(row*[integer]*)

Description

Returns x and y data for a point in a curve

Arguments

- **row** (integer)

The row point you want the data for. **Note that curve points start at 0, not 1**.

Returns

An array containing the x coordinate and the y coordinate.

Return type

Array

Example

To get the curve data for the 3rd point for curve l:

```
if (l.npoints >= 3)
{
    var point_data = l.GetPoint(2);
}
```

GetTableEntry(row*[integer]*)

Description

Returns the value and curve label for a row in a table

Arguments

- **row** (integer)

The row point you want the data for. **Note that curve points start at 0, not 1**.

Returns

An array containing the value and the load curve label.

Return type

Array

Example

To get the data for the 3rd point for table t:

```
if (t.npoints >= 3)
{
    var row_data = t.GetTableEntry(2);
}
```

InsertPoint(ipt*[integer]*, xvalue*[real]*, yvalue*[real]*, position*[integer]*)

Description

Inserts point values before or after a specified row of data on a load curve.

Arguments

- **ipt** (integer)

The row you want to insert the data before or after. **Note that the row data starts at 0, not 1**.

- **xvalue** (real)

The x value of the point.

- **yvalue** (real)

The y value of the point.

- **position** (integer)

Specify either before or after the selected row. Use ’Curve.BEFORE’ for before, and ’Curve.AFTER’ for after.

Returns

No return value.

Example

To insert the values after the 3rd row to x=3, y=5 for curve l:

```
l.InsertPoint(2, 3, 5, Curve.AFTER);
```

InsertTableEntry(ipt*[integer]*, value*[real]*, lcid*[integer]*, position*[integer]*)

Description

Inserts a table row before or after a specified row of data on a table.

Arguments

- **ipt** (integer)

The row you want to insert the data before or after. **Note that the row data starts at 0, not 1**.

- **value** (real)

The value of the row.

- **lcid** (integer)

The load curve corresponding to the defined value.

- **position** (integer)

Specify either before or after the selected row. Use ’Curve.BEFORE’ for before, and ’Curve.AFTER’ for after.

Returns

No return value.

Example

To insert the values after the 3rd row to value=3, lcur=5 for table t:

```
t.InsertTableEntry(2, 3, 5, Curve.AFTER);
```

Keyword()

Description

Returns the keyword for this curve (*DEFINE_CURVE_xxxx). **Note that a carriage return is not added**. See also Curve.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for curve l:

```
var key = l.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the curve. **Note that a carriage return is not added**. See also Curve.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for curve l:

```
var cards = l.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last curve in the model.

Arguments

- **Model** (Model)

Model to get last curve in

Returns

Curve object (or null if there are no curves in the model).

Return type

Curve

Example

To get the last curve in model m:

```
var c = Curve.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free curve label in the model. Also see Curve.FirstFreeLabel(), Curve.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free curve label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

Curve label.

Return type

Number

Example

To get the last free curve label in model m:

```
var label = Curve.LastFreeLabel(m);
```

Next()

Description

Returns the next curve in the model.

Arguments

No arguments

Returns

Curve object (or null if there are no more curves in the model).

Return type

Curve

Example

To get the curve in model m after curve c:

```
var c = c.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) curve label in the model. Also see Curve.FirstFreeLabel(), Curve.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free curve label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

Curve label.

Return type

Number

Example

To get the next free curve label in model m:

```
var label = Curve.NextFreeLabel(m);
```

Previous()

Description

Returns the previous curve in the model.

Arguments

No arguments

Returns

Curve object (or null if there are no more curves in the model).

Return type

Curve

Example

To get the curve in model m before curve c:

```
var c = c.Previous();
```

RemovePoint(row*[integer]*)

Description

Removes a row of data from a curve

Arguments

- **row** (integer)

The row point you want to remove. **Note that curve points start at 0, not 1**.

Returns

No return value.

Example

To remove the curve data for the 3rd point for curve l:

```
if (l.npoints >= 3)
{
    var point_data = l.RemovePoint(2);
}
```

RemoveTableEntry(ipt*[integer]*)

Description

Removes the value and loadcurve values for a specified row of data on a load curve.

Arguments

- **ipt** (integer)

The row you want to remove the data for. **Note that the row data starts at 0, not 1**.

Returns

No return value.

Example

To remove an entry at row 4:

```
t.RemoveTableEntry(4);
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the curves in the model.

Arguments

- **Model** (Model)

Model that all curves will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the curves in model m, from 1000000:

```
Curve.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged curves in the model.

Arguments

- **Model** (Model)

Model that all the flagged curves will be renumbered in

- **flag** (Flag)

Flag set on the curves that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the curves in model m flagged with f, from 1000000:

```
Curve.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select curves using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting curves

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only curves from that model can be selected. If the argument is a Flag then only curves that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any curves can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of curves selected or null if menu cancelled

Return type

Number

Example

To select curves from model m, flagging those selected with flag f, giving the prompt 'Select curves':

```
Curve.Select(f, 'Select curves', m);
```

To select curves, flagging those selected with flag f but limiting selection to curves flagged with flag l, giving the prompt 'Select curves':

```
Curve.Select(f, 'Select curves', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the curve.

Arguments

- **flag** (Flag)

Flag to set on the curve

Returns

No return value

Example

To set flag f for curve c:

```
c.SetFlag(f);
```

SetPoint(ipt*[integer]*, xvalue*[real]*, yvalue*[real]*)

Description

Sets the x and y values for a specified row of data on a load curve.

Arguments

- **ipt** (integer)

The row you want to set the data for. **Note that the row data starts at 0, not 1**.

- **xvalue** (real)

The x value of the point.

- **yvalue** (real)

The y value of the point.

Returns

No return value.

Example

To set the values for the 3rd row to x=3, y=5 for curve l:

```
l.SetPoint(2, 3, 5);
```

SetTableEntry(ipt*[integer]*, value*[real]*, load curve*[integer]*)

Description

Sets the value and loadcurve values for a specified row of data on a load curve.

Arguments

- **ipt** (integer)

The row you want to set the data for. **Note that the row data starts at 0, not 1**.

- **value** (real)

The value for for this entry in the table.

- **load curve** (integer)

The load curve corresponding to the defined value.

Returns

No return value.

Example

To add an entry with a value of 3 for load curve 1000 at row 4:

```
t.SetTableEntry(4, 3, 1000);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of curves in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing curves should be counted. If false or omitted referenced but undefined curves will also be included in the total.

Returns

number of curves

Return type

Number

Example

To get the total number of curves in model m:

```
var total = Curve.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the curves in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all curves will be unset in

- **flag** (Flag)

Flag to unset on the curves

Returns

No return value

Example

To unset the flag f on all the curves in model m:

```
Curve.UnflagAll(m, f);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

Curve object.

Return type

Curve

Example

To check if Curve property c.example is a parameter by using the Curve.GetParameter() method:

```
if (c.ViewParameters().GetParameter(c.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for curve with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The curve object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)

details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for curve c:

```
c.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for curve c:

```
c.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for curve. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for curve c:

```
c.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this curve.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for curve c:

```
var xrefs = c.Xrefs();
```

---

toString()

Description

Creates a string containing the curve data in keyword format. Note that this contains the keyword header and the keyword cards. See also Curve.Keyword() and Curve.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for curve l in keyword format

```
var l = d.toString();
```

ElementDeath class

The ElementDeath class gives you access to define element death cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

ElementDeath constants

Name	Description
ElementDeath.BEAM	Beam option
ElementDeath.BEAM_SET	Beam set option
ElementDeath.SHELL	Shell option
ElementDeath.SHELL_SET	Shell set option
ElementDeath.SOLID	Solid option

ElementDeath.SOLID_SET	Solid set option
ElementDeath.THICK_SHELL	Thick shell option
ElementDeath.THICK_SHELL_SET	Thick shell set option

ElementDeath properties

Name	Type	Description
boxid	integer	Box restricting element deletion
cid	integer	Coordinate ID for transforming boxid.
eid	integer	Element ID or element set ID. The sid property is an alternative name for this.
exists (read only)	logical	true if element death exists, false if referred to but not defined.
idgrp	integer	Group ID for simultaneous deletion.
include	integer	The Include file number that the element death is in.
inout	logical	If true, Ansys LS-DYNA deletes elements outside box, otherwise inside box.
model (read only)	integer	The Model number that the element death is in.
option	constant	ElementDeath option. Can be ElementDeath.SOLID, ElementDeath.SOLID_SET, ElementDeath.BEAM, ElementDeath.BEAM_SET, ElementDeath.SHELL, ElementDeath.SHELL_SET, ElementDeath.THICK_SHELL or ElementDeath.THICK_SHELL_SET. The type property is an alternative name for this.
percent	real	Deletion percentage.
sid	integer	Element ID or element set ID. The eid property is an alternative name for this.
time	real	Deletion time for elimination
title	string	ElementDeath title
type	constant	ElementDeath option. Can be ElementDeath.SOLID, ElementDeath.SOLID_SET, ElementDeath.BEAM, ElementDeath.BEAM_SET, ElementDeath.SHELL, ElementDeath.SHELL_SET, ElementDeath.THICK_SHELL or ElementDeath.THICK_SHELL_SET The option property is an alternative name for this.

Detailed Description

The ElementDeath class allows you to create, modify, edit and manipulate element death cards. See the documentation below for more details.

Constructor

new ElementDeath(Model*[Model]*, type*[string]*, eid/sid*[integer]*)

Description

Create a new ElementDeath object.

Arguments

- **Model** (Model)

Model that element death will be created in

- **type** (string)

ElementDeath type. Can be ElementDeath.SOLID, ElementDeath.SOLID_SET, ElementDeath.BEAM,

ElementDeath.BEAM_SET, ElementDeath.SHELL, ElementDeath.SHELL_SET, ElementDeath.THICK_SHELL or ElementDeath.THICK_SHELL_SET

- **eid/sid** (integer)

Element or element set ID

Returns

ElementDeath object

Return type

ElementDeath

Example

To create a new element death in model m with option BEAM_SET and sid 100

```
var ed = new ElementDeath(m, ElementDeath.BEAM_SET, 100);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a element death.

Arguments

- **Comment** (Comment)

Comment that will be attached to the element death

Returns

No return value

Example

To associate comment c to the element death ed:

```
ed.AssociateComment(c);
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse element death ed:

```
ed.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the element death.

Arguments

- **flag** (Flag)

Flag to clear on the element death

Returns

No return value

Example

To clear flag f for element death ed:

```
ed.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the element death. The target include of the copied element death can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

ElementDeath object

Return type

ElementDeath

Example

To copy element death ed into element death z:

```
var z = ed.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a element death

Arguments

- **Model** (Model)

Model that the element death will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

ElementDeath object (or null if not made).

Return type

ElementDeath

Example

To start creating a element death in model m:

```
var ed = ElementDeath.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a element death.

Arguments

- **Comment** (Comment)

Comment that will be detached from the element death

Returns

No return value

Example

To detach comment c from the element death ed:

```
ed.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit element death ed:

```
ed.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for element death with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The element death object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for element death ed:

```
ed.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for element death ed:

```
ed.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for element death. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for element death ed:

```
ed.Error("My custom error");
```

First(Model[*Model*]) [static]

Description

Returns the first element death in the model.

Arguments

- **Model** (Model)

Model to get first element death in

Returns

ElementDeath object (or null if there are no element deaths in the model).

Return type

ElementDeath

Example

To get the first element death in model m:

```
var ed = ElementDeath.First(m);
```

FlagAll(Model[*Model*], flag[*Flag*]) [static]

Description

Flags all of the element deaths in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all element deaths will be flagged in

- **flag** (Flag)

Flag to set on the element deaths

Returns

No return value

Example

To flag all of the element deaths with flag f in model m:

```
ElementDeath.FlagAll(m, f);
```

Flagged(flag[*Flag*])

Description

Checks if the element death is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the element death

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if element death ed has flag f set on it:

```
if (ed.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each element death in the model.
Note that ForEach has been designed to make looping over element deaths as fast as possible and so has some limitations.
Firstly, a single temporary ElementDeath object is created and on each function call it is updated with the current element death data. This means that you should not try to store the ElementDeath object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new element deaths inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all element deaths are in

- **func** (function)

Function to call for each element death

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the element deaths in model m:

```
ElementDeath.ForEach(m, test);
function test(ed)
{
// ed is ElementDeath object
}
```

To call function test for all of the element deaths in model m with optional object:

```
var data = { x:0, y:0 };
ElementDeath.ForEach(m, test, data);
function test(ed, extra)
{
// ed is ElementDeath object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of ElementDeath objects or properties for all of the element deaths in a model in PRIMER. If the optional property argument is not given then an array of ElementDeath objects is returned. If the property argument is given, that property value for each element death is returned in the array instead of a ElementDeath object

Arguments

- **Model** (Model)

Model to get element deaths from

- **property (optional)** (string)

Name for property to get for all element deaths in the model

Returns

Array of ElementDeath objects or properties

Return type

Array

Example

To make an array of ElementDeath objects for all of the element deaths in model m:

```
var a = ElementDeath.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each element death in model m:

```
var a = ElementDeath.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a element death.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the element death ed:

```
var comm_array = ed.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of ElementDeath objects for all of the flagged element deaths in a model in PRIMER If the optional property argument is not given then an array of ElementDeath objects is returned. If the property argument is given, then that property value for each element death is returned in the array instead of a ElementDeath object

Arguments

- **Model** (Model)

Model to get element deaths from

- **flag** (Flag)

Flag set on the element deaths that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged element deaths in the model

Returns

Array of ElementDeath objects or properties

Return type

Array

Example

To make an array of ElementDeath objects for all of the element deaths in model m flagged with f:

```
var ed = ElementDeath.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the element deaths in model m flagged with f:

```
var a = ElementDeath.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the ElementDeath object for a element death ID.

Arguments

- **Model** (Model)

Model to find the element death in

- **number** (integer)

number of the element death you want the ElementDeath object for

Returns

ElementDeath object (or null if element death does not exist).

Return type

ElementDeath

Example

To get the ElementDeath object for element death 100 in model m

```
var ed = ElementDeath.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a ElementDeath property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the ElementDeath.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

element death property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if ElementDeath property ed.example is a parameter:

```
Options.property_parameter_names = true;
if (ed.GetParameter(ed.example) ) do_something...
Options.property_parameter_names = false;
```

To check if ElementDeath property ed.example is a parameter by using the GetParameter method:

```
if (ed.ViewParameters().GetParameter(ed.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this element death (*DEFINE_ELEMENT_DEATH). **Note that a carriage return is not added**. See also ElementDeath.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for element death ed:

```
var key = ed.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the element death. **Note that a carriage return is not added**. See also ElementDeath.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for element death ed:

```
var cards = ed.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last element death in the model.

Arguments

- **Model** (Model)

Model to get last element death in

Returns

ElementDeath object (or null if there are no element deaths in the model).

Return type

ElementDeath

Example

To get the last element death in model m:

```
var ed = ElementDeath.Last(m);
```

Next()

Description

Returns the next element death in the model.

Arguments

No arguments

Returns

ElementDeath object (or null if there are no more element deaths in the model).

Return type

ElementDeath

Example

To get the element death in model m after element death ed:

```
var ed = ed.Next();
```

Previous()

Description

Returns the previous element death in the model.

Arguments

No arguments

Returns

ElementDeath object (or null if there are no more element deaths in the model).

Return type

ElementDeath

Example

To get the element death in model m before element death ed:

```
var ed = ed.Previous();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select element deaths using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting element deaths

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only element deaths from that model can be selected. If the argument is a Flag then only element deaths that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any element deaths can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of element deaths selected or null if menu cancelled

Return type

Number

Example

To select element deaths from model m, flagging those selected with flag f, giving the prompt 'Select element deaths':

```
ElementDeath.Select(f, 'Select element deaths', m);
```

To select element deaths, flagging those selected with flag f but limiting selection to element deaths flagged with flag l, giving the prompt 'Select element deaths':

```
ElementDeath.Select(f, 'Select element deaths', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the element death.

Arguments

- **flag** (Flag)

Flag to set on the element death

Returns

No return value

Example

To set flag f for element death ed:

```
ed.SetFlag(f);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of element deaths in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing element deaths should be counted. If false or omitted referenced but undefined element deaths will also be included in the total.

Returns

number of element deaths

Return type

Number

Example

To get the total number of element deaths in model m:

```
var total = ElementDeath.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the element deaths in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all element deaths will be unset in

- **flag** (Flag)

Flag to unset on the element deaths

Returns

No return value

Example

To unset the flag f on all the element deaths in model m:

```
ElementDeath.UnflagAll(m, f);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

ElementDeath object.

Return type

ElementDeath

Example

To check if ElementDeath property ed.example is a parameter by using the ElementDeath.GetParameter() method:

```
if (ed.ViewParameters().GetParameter(ed.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for element death with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The element death object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for element death ed:

```
ed.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for element death ed:

```
ed.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for element death. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for element death ed:

```
ed.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this element death.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for element death ed:

```
var xrefs = ed.Xrefs();
```

---

toString()

Description

Creates a string containing the element death data in keyword format. Note that this contains the keyword header and the keyword cards. See also ElementDeath.Keyword() and ElementDeath.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for element death ed in keyword format

```
var s = ed.toString();
```

HexSpotweldAssembly class

The HexSpotweldAssembly class gives you access to *DEFINE_HEX_SPOTWELD_ASSEMBLY cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

HexSpotweldAssembly properties

Name	Type	Description
eid1	integer	EID 1
eid10	integer	EID 10

eid11	integer	EID 11
eid12	integer	EID 12
eid13	integer	EID 13
eid14	integer	EID 14
eid15	integer	EID 15
eid16	integer	EID 16
eid2	integer	EID 2
eid3	integer	EID 3
eid4	integer	EID 4
eid5	integer	EID 5
eid6	integer	EID 6
eid7	integer	EID 7
eid8	integer	EID 8
eid9	integer	EID 9
exists (read only)	logical	true if DEFINE_HEX_SPOTWELD_ASSEMBLY exists, false if referred to but not defined.
id	integer	*DEFINE_HEX_SPOTWELD_ASSEMBLY id
include	integer	The Include file number that the DEFINE_HEX_SPOTWELD_ASSEMBLY is in.
model (read only)	integer	The Model number that the DEFINE_HEX_SPOTWELD_ASSEMBLY is in.
opt	integer	*DEFINE_HEX_SPOTWELD_ASSEMBLY opt
title	string	Title (optional)

Detailed Description

The HexSpotweldAssembly class allows you to create, modify, edit and manipulate *DEFINE_HEX_SPOTWELD_ASSEMBLY cards. See the documentation below for more details.

Constructor

new HexSpotweldAssembly(Model*[Model]*, options *[object]*)

Description

Create a new *DEFINE_HEX_SPOTWELD_ASSEMBLY object.

Arguments

- **Model** (Model)

Model that Hex Spotweld Assembly will be created in

- **options** (object)

Options for creating the HexSpotweldAssembly

Object has the following properties:

Name	Type	Description
id	integer	HexSpotweldAssembly ID.

opt	integer	HexSpotweldAssembly option indicating the length of the solids array. opt can be 4, 8 or 16.
solids	Array of numbers	Array of Solid IDs, at least 4 EIDs must be given.
title (optional)	string	Optional HexSpotweldAssembly title.

Returns

HexSpotweldAssembly object

Return type

HexSpotweldAssembly

Example

To create a new *DEFINE_HEX_SPOTWELD_ASSEMBLY with ID 100 in model m with 4 elements 50, 150, 250 and 350

```
var h = new HexSpotweldAssembly(m, {id: 100, opt: 4, solids: [50, 150, 250, 350]});
```

To create a new *DEFINE_HEX_SPOTWELD_ASSEMBLY with ID 200 in model m with 8 elements 50, 150, 250, 350, 450, 550, 650 and 750

```
var h = new HexSpotweldAssembly(m, {id: 200, opt: 8, solids: [50, 150, 250, 350, 450, 550, 650, 750]});
```

To create a new *DEFINE_HEX_SPOTWELD_ASSEMBLY with ID 300 in model m with 16 elements 50, 150, 250, 350, 450, 550, 650, 750, 850, 950, 1050, 1150, 1250, 1350, 1450 and 1550

```
var h = new HexSpotweldAssembly(m, {id: 300, opt: 16, solids: [50, 150, 250, 350, 450, 550, 650, 750, 850, 950, 1050, 1150, 1250, 1350, 1450, 1550]});
```

new HexSpotweldAssembly(Model*[Model]*, id*[integer]*, opt*[integer]*, eid1*[integer]*, eid2*[integer]*, eid3*[integer]*, eid4*[integer]*, title (optional)*[string]*) [deprecated]

This function is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Create a new *DEFINE_HEX_SPOTWELD_ASSEMBLY object.

Arguments

- **Model** (Model)

Model that Hex Spotweld Assembly will be created in

- **id** (integer)

*DEFINE_HEX_SPOTWELD_ASSEMBLY id_sw.

- **opt** (integer)

*DEFINE_HEX_SPOTWELD_ASSEMBLY opt can be 4, 8 or 16

- **eid1** (integer)

EID 1.

- **eid2** (integer)

EID 2.

- **eid3** (integer)

EID 3.

- **eid4** (integer)

EID 4.

- **title (optional)** (string)

Define hex spotweld assembly title.

Returns

HexSpotweldAssembly object

Return type

HexSpotweldAssembly

Example

To create a new *DEFINE_HEX_SPOTWELD_ASSEMBLY with ID 100 in model m with 4 elements 50, 150, 250 and 350

```
var h = new HexSpotweldAssembly(m, 100, 4, 50, 150, 250, 350);
```

new HexSpotweldAssembly(Model*[Model]*, id*[integer]*, opt*[integer]*, eid1*[integer]*, eid2*[integer]*, eid3*[integer]*, eid4*[integer]*, eid5*[integer]*, eid6*[integer]*, eid7*[integer]*, eid8*[integer]*, title (optional)*[string]*) [deprecated]

This function is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Create a new *DEFINE_HEX_SPOTWELD_ASSEMBLY object.

Arguments

- **Model** (Model)

Model that Hex Spotweld Assembly will be created in

- **id** (integer)

*DEFINE_HEX_SPOTWELD_ASSEMBLY id.

- **opt** (integer)

*DEFINE_HEX_SPOTWELD_ASSEMBLY opt can be 4, 8 or 16

- **eid1** (integer)

EID 1.

- **eid2** (integer)

EID 2.

- **eid3** (integer)

EID 3.

- **eid4** (integer)

EID 4.

- **eid5** (integer)

EID 5.

- **eid6** (integer)

EID 6.

- **eid7** (integer)

EID 7.

- **eid8** (integer)

EID 8.

- **title (optional)** (string)

Define hex spotweld assembly title.

Returns

HexSpotweldAssembly object

Return type

HexSpotweldAssembly

Example

To create a new *DEFINE_HEX_SPOTWELD_ASSEMBLY with ID 100 in model m with 8 elements 50, 150, 250, 350, 450, 550, 650 and 750

```
var h = new HexSpotweldAssembly(m, 100, 8, 50, 150, 250, 350, 450, 550, 650, 750);
```

new HexSpotweldAssembly(Model*[Model]*, id*[integer]*, opt*[integer]*, eid1*[integer]*, eid2*[integer]*, eid3*[integer]*, eid4*[integer]*, eid5*[integer]*, eid6*[integer]*, eid7*[integer]*, eid8*[integer]*, eid9*[integer]*, eid10*[integer]*, eid11*[integer]*, eid12*[integer]*, eid13*[integer]*, eid14*[integer]*, eid15*[integer]*, eid16*[integer]*, title (optional)*[string]*) [deprecated]

This function is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Create a new *DEFINE_HEX_SPOTWELD_ASSEMBLY object.

Arguments

- **Model** (Model)

Model that Hex Spotweld Assembly will be created in

- **id** (integer)

*DEFINE_HEX_SPOTWELD_ASSEMBLY id.

- **opt** (integer)

*DEFINE_HEX_SPOTWELD_ASSEMBLY opt can be 4, 8 or 16

- **eid1** (integer)

EID 1.

- **eid2** (integer)

EID 2.

- **eid3** (integer)

EID 3.

- **eid4** (integer)

EID 4.

- **eid5** (integer)

EID 5.

- **eid6** (integer)

EID 6.

- **eid7** (integer)

EID 7.

- **eid8** (integer)

EID 8.

- **eid9** (integer)

EID 9.

- **eid10** (integer)

EID 10.

- **eid11** (integer)

EID 11.

- **eid12** (integer)

EID 12.

- **eid13** (integer)

EID 13.

- **eid14** (integer)

EID 14.

- **eid15** (integer)

EID 15.

- **eid16** (integer)

EID 16.

- **title (optional)** (string)

Define hex spotweld assembly title.

Returns

HexSpotweldAssembly object

Return type

HexSpotweldAssembly

Example

To create a new *DEFINE_HEX_SPOTWELD_ASSEMBLY with ID 100 in model m with 16 elements 50, 150, 250, 350, 450, 550, 650, 750, 850, 950, 1050, 1150, 1250, 1350, 1450 and 1550

```
var h = new HexSpotweldAssembly(m, 100, 16, 50, 150, 250, 350, 450, 550, 650,
750, 850, 950, 1050, 1150, 1250, 1350, 1450, 1550);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a DEFINE_HEX_SPOTWELD_ASSEMBLY.

Arguments

- **Comment** (Comment)

Comment that will be attached to the DEFINE_HEX_SPOTWELD_ASSEMBLY

Returns

No return value

Example

To associate comment c to the DEFINE_HEX_SPOTWELD_ASSEMBLY h:

```
h.AssociateComment(c);
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse DEFINE_HEX_SPOTWELD_ASSEMBLY h:

```
h.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the DEFINE_HEX_SPOTWELD_ASSEMBLY.

Arguments

- **flag** (Flag)

Flag to clear on the DEFINE_HEX_SPOTWELD_ASSEMBLY

Returns

No return value

Example

To clear flag f for DEFINE_HEX_SPOTWELD_ASSEMBLY h:

```
h.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the DEFINE_HEX_SPOTWELD_ASSEMBLY. The target include of the copied DEFINE_HEX_SPOTWELD_ASSEMBLY can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

HexSpotweldAssembly object

Return type

HexSpotweldAssembly

Example

To copy DEFINE_HEX_SPOTWELD_ASSEMBLY h into DEFINE_HEX_SPOTWELD_ASSEMBLY z:

```
var z = h.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a DEFINE_HEX_SPOTWELD_ASSEMBLY

Arguments

- **Model** (Model)

Model that the DEFINE_HEX_SPOTWELD_ASSEMBLY will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

HexSpotweldAssembly object (or null if not made).

Return type

HexSpotweldAssembly

Example

To start creating a DEFINE_HEX_SPOTWELD_ASSEMBLY in model m:

```
var h = HexSpotweldAssembly.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a DEFINE_HEX_SPOTWELD_ASSEMBLY.

Arguments

- **Comment** (Comment)

Comment that will be detached from the DEFINE_HEX_SPOTWELD_ASSEMBLY

Returns

No return value

Example

To detach comment c from the DEFINE_HEX_SPOTWELD_ASSEMBLY h:

```
h.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit DEFINE_HEX_SPOTWELD_ASSEMBLY h:

```
h.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for DEFINE_HEX_SPOTWELD_ASSEMBLY with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The DEFINE_HEX_SPOTWELD_ASSEMBLY object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for DEFINE_HEX_SPOTWELD_ASSEMBLY h:

```
h.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for DEFINE_HEX_SPOTWELD_ASSEMBLY h:

```
h.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for DEFINE_HEX_SPOTWELD_ASSEMBLY. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for DEFINE_HEX_SPOTWELD_ASSEMBLY h:

```
h.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first DEFINE_HEX_SPOTWELD_ASSEMBLY in the model.

Arguments

- **Model** (Model)

Model to get first DEFINE_HEX_SPOTWELD_ASSEMBLY in

Returns

HexSpotweldAssembly object (or null if there are no DEFINE_HEX_SPOTWELD_ASSEMBLYs in the model).

Return type

HexSpotweldAssembly

Example

To get the first DEFINE_HEX_SPOTWELD_ASSEMBLY in model m:

```
var h = HexSpotweldAssembly.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free DEFINE_HEX_SPOTWELD_ASSEMBLY label in the model. Also see HexSpotweldAssembly.LastFreeLabel(), HexSpotweldAssembly.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free DEFINE_HEX_SPOTWELD_ASSEMBLY label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

HexSpotweldAssembly label.

Return type

Number

Example

To get the first free DEFINE_HEX_SPOTWELD_ASSEMBLY label in model m:

```
var label = HexSpotweldAssembly.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the DEFINE_HEX_SPOTWELD_ASSEMBLYs in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all DEFINE_HEX_SPOTWELD_ASSEMBLYs will be flagged in

- **flag** (Flag)

Flag to set on the DEFINE_HEX_SPOTWELD_ASSEMBLYs

Returns

No return value

Example

To flag all of the DEFINE_HEX_SPOTWELD_ASSEMBLYs with flag f in model m:

```
HexSpotweldAssembly.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the DEFINE_HEX_SPOTWELD_ASSEMBLY is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the DEFINE_HEX_SPOTWELD_ASSEMBLY

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if DEFINE_HEX_SPOTWELD_ASSEMBLY h has flag f set on it:

```
if (h.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each DEFINE_HEX_SPOTWELD_ASSEMBLY in the model.
Note that ForEach has been designed to make looping over DEFINE_HEX_SPOTWELD_ASSEMBLYs as fast as possible and so has some limitations.
Firstly, a single temporary HexSpotweldAssembly object is created and on each function call it is updated with the current DEFINE_HEX_SPOTWELD_ASSEMBLY data. This means that you should not try to store the HexSpotweldAssembly object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new DEFINE_HEX_SPOTWELD_ASSEMBLYs inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all DEFINE_HEX_SPOTWELD_ASSEMBLYs are in

- **func** (function)

Function to call for each DEFINE_HEX_SPOTWELD_ASSEMBLY

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the DEFINE_HEX_SPOTWELD_ASSEMBLYs in model m:

```
HexSpotweldAssembly.ForEach(m, test);
function test(h)
{
// h is HexSpotweldAssembly object
}
```

To call function test for all of the DEFINE_HEX_SPOTWELD_ASSEMBLYs in model m with optional object:

```
var data = { x:0, y:0 };
HexSpotweldAssembly.ForEach(m, test, data);
function test(h, extra)
{
// h is HexSpotweldAssembly object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of HexSpotweldAssembly objects or properties for all of the DEFINE_HEX_SPOTWELD_ ASSEMBLYs in a model in PRIMER. If the optional property argument is not given then an array of HexSpotweldAssembly objects is returned. If the property argument is given, that property value for each DEFINE_ HEX_SPOTWELD_ASSEMBLY is returned in the array instead of a HexSpotweldAssembly object

Arguments

- **Model** (Model)

Model to get DEFINE_HEX_SPOTWELD_ASSEMBLYs from

- **property (optional)** (string)

Name for property to get for all DEFINE_HEX_SPOTWELD_ASSEMBLYs in the model

Returns

Array of HexSpotweldAssembly objects or properties

Return type

Array

Example

To make an array of HexSpotweldAssembly objects for all of the DEFINE_HEX_SPOTWELD_ASSEMBLYs in model m:

```
var a = HexSpotweldAssembly.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each DEFINE_HEX_SPOTWELD_ASSEMBLY in model m:

```
var a = HexSpotweldAssembly.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a DEFINE_HEX_SPOTWELD_ASSEMBLY.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the DEFINE_HEX_SPOTWELD_ASSEMBLY h:

```
var comm_array = h.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of HexSpotweldAssembly objects for all of the flagged DEFINE_HEX_SPOTWELD_ASSEMBLYs in a model in PRIMER If the optional property argument is not given then an array of HexSpotweldAssembly objects is returned. If the property argument is given, then that property value for each DEFINE_HEX_SPOTWELD_ASSEMBLY is returned in the array instead of a HexSpotweldAssembly object

Arguments

- **Model** (Model)

Model to get DEFINE_HEX_SPOTWELD_ASSEMBLYs from

- **flag** (Flag)

Flag set on the DEFINE_HEX_SPOTWELD_ASSEMBLYs that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged DEFINE_HEX_SPOTWELD_ASSEMBLYs in the model

Returns

Array of HexSpotweldAssembly objects or properties

Return type

Array

Example

To make an array of HexSpotweldAssembly objects for all of the DEFINE_HEX_SPOTWELD_ASSEMBLYs in model m flagged with f:

```
var h = HexSpotweldAssembly.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the DEFINE_HEX_SPOTWELD_ASSEMBLYs in model m flagged with f:

```
var a = HexSpotweldAssembly.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the HexSpotweldAssembly object for a DEFINE_HEX_SPOTWELD_ASSEMBLY ID.

Arguments

- **Model** (Model)

Model to find the DEFINE_HEX_SPOTWELD_ASSEMBLY in

- **number** (integer)

number of the DEFINE_HEX_SPOTWELD_ASSEMBLY you want the HexSpotweldAssembly object for

Returns

HexSpotweldAssembly object (or null if DEFINE_HEX_SPOTWELD_ASSEMBLY does not exist).

Return type

HexSpotweldAssembly

Example

To get the HexSpotweldAssembly object for DEFINE_HEX_SPOTWELD_ASSEMBLY 100 in model m

```
var h = HexSpotweldAssembly.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a HexSpotweldAssembly property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the HexSpotweldAssembly.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

DEFINE_HEX_SPOTWELD_ASSEMBLY property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if HexSpotweldAssembly property h.example is a parameter:

```
Options.property_parameter_names = true;
if (h.GetParameter(h.example) ) do_something...
Options.property_parameter_names = false;
```

To check if HexSpotweldAssembly property h.example is a parameter by using the GetParameter method:

```
if (h.ViewParameters().GetParameter(h.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this HexSpotweldAssembly (*DEFINE_HEX_SPOTWELD_ASSEMBLY). **Note that a carriage return is not added**. See also HexSpotweldAssembly.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for HexSpotweldAssem s:

```
var key = s.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the HexSpotweldAssem. **Note that a carriage return is not added**. See also HexSpotweldAssembly.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for HexSpotweldAssem s:

```
var cards = s.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last DEFINE_HEX_SPOTWELD_ASSEMBLY in the model.

Arguments

- **Model** (Model)

Model to get last DEFINE_HEX_SPOTWELD_ASSEMBLY in

Returns

HexSpotweldAssembly object (or null if there are no DEFINE_HEX_SPOTWELD_ASSEMBLYs in the model).

Return type

HexSpotweldAssembly

Example

To get the last DEFINE_HEX_SPOTWELD_ASSEMBLY in model m:

```
var h = HexSpotweldAssembly.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free DEFINE_HEX_SPOTWELD_ASSEMBLY label in the model. Also see HexSpotweldAssembly.FirstFreeLabel(), HexSpotweldAssembly.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free DEFINE_HEX_SPOTWELD_ASSEMBLY label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

HexSpotweldAssembly label.

Return type

Number

Example

To get the last free DEFINE_HEX_SPOTWELD_ASSEMBLY label in model m:

```
var label = HexSpotweldAssembly.LastFreeLabel(m);
```

Next()

Description

Returns the next DEFINE_HEX_SPOTWELD_ASSEMBLY in the model.

Arguments

No arguments

Returns

HexSpotweldAssembly object (or null if there are no more DEFINE_HEX_SPOTWELD_ASSEMBLYs in the model).

Return type

HexSpotweldAssembly

Example

To get the DEFINE_HEX_SPOTWELD_ASSEMBLY in model m after DEFINE_HEX_SPOTWELD_ASSEMBLY h:

```
var h = h.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) DEFINE_HEX_SPOTWELD_ASSEMBLY label in the model. Also see HexSpotweldAssembly.FirstFreeLabel(), HexSpotweldAssembly.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free DEFINE_HEX_SPOTWELD_ASSEMBLY label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

HexSpotweldAssembly label.

Return type

Number

Example

To get the next free DEFINE_HEX_SPOTWELD_ASSEMBLY label in model m:

```
var label = HexSpotweldAssembly.NextFreeLabel(m);
```

Previous()

Description

Returns the previous DEFINE_HEX_SPOTWELD_ASSEMBLY in the model.

Arguments

No arguments

Returns

HexSpotweldAssembly object (or null if there are no more DEFINE_HEX_SPOTWELD_ASSEMBLYs in the model).

Return type

HexSpotweldAssembly

Example

To get the DEFINE_HEX_SPOTWELD_ASSEMBLY in model m before DEFINE_HEX_SPOTWELD_ASSEMBLY h:

```
var h = h.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the DEFINE_HEX_SPOTWELD_ASSEMBLYs in the model.

Arguments

- **Model** (Model)

Model that all DEFINE_HEX_SPOTWELD_ASSEMBLYs will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the DEFINE_HEX_SPOTWELD_ASSEMBLYs in model m, from 1000000:

```
HexSpotweldAssembly.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged DEFINE_HEX_SPOTWELD_ASSEMBLYs in the model.

Arguments

- **Model** (Model)

Model that all the flagged DEFINE_HEX_SPOTWELD_ASSEMBLYs will be renumbered in

- **flag** (Flag)

Flag set on the DEFINE_HEX_SPOTWELD_ASSEMBLYs that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the DEFINE_HEX_SPOTWELD_ASSEMBLYs in model m flagged with f, from 1000000:

```
HexSpotweldAssembly.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select DEFINE_HEX_SPOTWELD_ASSEMBLYs using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting DEFINE_HEX_SPOTWELD_ASSEMBLYs

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only DEFINE_HEX_SPOTWELD_ASSEMBLYs from that model can be selected. If the argument is a Flag then only DEFINE_HEX_SPOTWELD_ASSEMBLYs that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any DEFINE_HEX_SPOTWELD_ASSEMBLYs can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of DEFINE_HEX_SPOTWELD_ASSEMBLYs selected or null if menu cancelled

Return type

Number

Example

To select DEFINE_HEX_SPOTWELD_ASSEMBLYs from model m, flagging those selected with flag f, giving the prompt 'Select DEFINE_HEX_SPOTWELD_ASSEMBLYs':

```
HexSpotweldAssembly.Select(f, 'Select DEFINE_HEX_SPOTWELD_ASSEMBLYs', m);
```

To select DEFINE_HEX_SPOTWELD_ASSEMBLYs, flagging those selected with flag f but limiting selection to DEFINE_HEX_SPOTWELD_ASSEMBLYs flagged with flag l, giving the prompt 'Select DEFINE_HEX_SPOTWELD_ASSEMBLYs':

```
HexSpotweldAssembly.Select(f, 'Select DEFINE_HEX_SPOTWELD_ASSEMBLYs', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the DEFINE_HEX_SPOTWELD_ASSEMBLY.

Arguments

- **flag** (Flag)

Flag to set on the DEFINE_HEX_SPOTWELD_ASSEMBLY

Returns

No return value

Example

To set flag f for DEFINE_HEX_SPOTWELD_ASSEMBLY h:

```
h.SetFlag(f);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of DEFINE_HEX_SPOTWELD_ASSEMBLYs in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing DEFINE_HEX_SPOTWELD_ASSEMBLYs should be counted. If false or omitted referenced but undefined DEFINE_HEX_SPOTWELD_ASSEMBLYs will also be included in the total.

Returns

number of DEFINE_HEX_SPOTWELD_ASSEMBLYs

Return type

Number

Example

To get the total number of DEFINE_HEX_SPOTWELD_ASSEMBLYs in model m:

```
var total = HexSpotweldAssembly.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the DEFINE_HEX_SPOTWELD_ASSEMBLYs in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all DEFINE_HEX_SPOTWELD_ASSEMBLYs will be unset in

- **flag** (Flag)

Flag to unset on the DEFINE_HEX_SPOTWELD_ASSEMBLYs

Returns

No return value

Example

To unset the flag f on all the DEFINE_HEX_SPOTWELD_ASSEMBLYs in model m:

```
HexSpotweldAssembly.UnflagAll(m, f);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

HexSpotweldAssembly object.

Return type

HexSpotweldAssembly

Example

To check if HexSpotweldAssembly property h.example is a parameter by using the HexSpotweldAssembly.GetParameter() method:

```
if (h.ViewParameters().GetParameter(h.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for DEFINE_HEX_SPOTWELD_ASSEMBLY with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The DEFINE_HEX_SPOTWELD_ASSEMBLY object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for DEFINE_HEX_SPOTWELD_ASSEMBLY h:

```
h.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for DEFINE_HEX_SPOTWELD_ASSEMBLY h:

```
h.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for DEFINE_HEX_SPOTWELD_ASSEMBLY. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for DEFINE_HEX_SPOTWELD_ASSEMBLY h:

```
h.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this DEFINE_HEX_SPOTWELD_ASSEMBLY.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for DEFINE_HEX_SPOTWELD_ASSEMBLY h:

```
var xrefs = h.Xrefs();
```

---

toString()

Description

Creates a string containing the HexSpotweldAssem data in keyword format. Note that this contains the keyword header and the keyword cards. See also HexSpotweldAssembly.Keyword() and HexSpotweldAssembly.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for HexSpotweldAssem s in keyword format

```
var str = s.toString();
```

StagedConstructionPart class

The StagedConstructionPart class gives you access to Define staged construction part cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

StagedConstructionPart constants

Name	Description
StagedConstructionPart.PART	DEFN is *DEFINE_STAGED_CONSTRUCTION_PART.
StagedConstructionPart.SET	DEFN is *DEFINE_STAGED_CONSTRUCTION_PART_SET.

StagedConstructionPart properties

Name	Type	Description
exists (read only)	logical	true if Define staged construction part exists, false if referred to but not defined.
id	integer	Part ID or part set ID (not internal label)
include	integer	The Include file number that the Define staged construction part is in.
label (read only)	integer	The label the Define staged construction parts has in PRIMER
model (read only)	integer	The Model number that the Define staged construction part is in.
option	constant	The Define staged construction parts option. Can be StagedConstructionPart.PART or StagedConstructionPart.SET.
stga	integer	Construction stage at which part is added.
stgr	integer	Construction stage at which part is removed.

Detailed Description

The StagedConstructionPart class allows you to create, modify, edit and manipulate Define staged construction parts cards. See the documentation below for more details.

Constructor

new StagedConstructionPart(Model*[Model]*, option*[constant]*, id*[integer]*, stga*[integer]*, stgr*[integer]*)

Description

Create a new StagedConstructionPart object.

Arguments

- **Model** (Model)

Model that Define staged construction parts will be created in

- **option** (constant)

Specify the type of Define staged construction parts. Can be StagedConstructionPart.PART or StagedConstructionPart.SET)

- **id** (integer)

Part ID or part set ID

- **stga** (integer)

Construction stage at which part is added.

- **stgr** (integer)

Construction stage at which part is removed.

Returns

StagedConstructionPart object

Return type

StagedConstructionPart

Example

To create a new Define staged construction part in model m, of type SET, with part set 9, stga 18 and stgr 12

```
var scp = new StagedConstructionPart(m, StagedConstructionPart.SET, 9, 18, 12);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a Define staged construction part.

Arguments

- **Comment** (Comment)

Comment that will be attached to the Define staged construction part

Returns

No return value

Example

To associate comment c to the Define staged construction part scp:

```
scp.AssociateComment(c);
```

Blank()

Description

Blanks the Define staged construction part

Arguments

No arguments

Returns

No return value

Example

To blank Define staged construction part scp:

```
scp.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the Define staged construction parts in the model.

Arguments

- **Model** (Model)

Model that all Define staged construction parts will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the Define staged construction parts in model m:

```
StagedConstructionPart.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged Define staged construction parts in the model.

Arguments

- **Model** (Model)

Model that all the flagged Define staged construction parts will be blanked in

- **flag** (Flag)

Flag set on the Define staged construction parts that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the Define staged construction parts in model m flagged with f:

```
StagedConstructionPart.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the Define staged construction part is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if Define staged construction part scp is blanked:

```
if (scp.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse Define staged construction part scp:

```
scp.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the Define staged construction part.

Arguments

- **flag** (Flag)

Flag to clear on the Define staged construction part

Returns

No return value

Example

To clear flag f for Define staged construction part scp:

```
scp.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the Define staged construction part. The target include of the copied Define staged construction part can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

StagedConstructionPart object

Return type

StagedConstructionPart

Example

To copy Define staged construction part scp into Define staged construction part z:

```
var z = scp.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a Define staged construction part

Arguments

- **Model** (Model)

Model that the Define staged construction part will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

StagedConstructionPart object (or null if not made).

Return type

StagedConstructionPart

Example

To start creating a Define staged construction part in model m:

```
var scp = StagedConstructionPart.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a Define staged construction part.

Arguments

- **Comment** (Comment)

Comment that will be detached from the Define staged construction part

Returns

No return value

Example

To detach comment c from the Define staged construction part scp:

```
scp.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit Define staged construction part scp:

```
scp.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for Define staged construction part with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The Define staged construction part object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for Define staged construction part scp:

```
scp.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for Define staged construction part scp:

```
scp.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for Define staged construction part. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for Define staged construction part scp:

```
scp.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first Define staged construction part in the model.

Arguments

- **Model** (Model)

Model to get first Define staged construction part in

Returns

StagedConstructionPart object (or null if there are no Define staged construction parts in the model).

Return type

StagedConstructionPart

Example

To get the first Define staged construction part in model m:

```
var scp = StagedConstructionPart.First(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the Define staged construction parts in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all Define staged construction parts will be flagged in

- **flag** (Flag)

Flag to set on the Define staged construction parts

Returns

No return value

Example

To flag all of the Define staged construction parts with flag f in model m:

```
StagedConstructionPart.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the Define staged construction part is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the Define staged construction part

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if Define staged construction part scp has flag f set on it:

```
if (scp.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each Define staged construction part in the model.
Note that ForEach has been designed to make looping over Define staged construction parts as fast as possible and so has some limitations.
Firstly, a single temporary StagedConstructionPart object is created and on each function call it is updated with the current Define staged construction part data. This means that you should not try to store the StagedConstructionPart object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new Define staged construction parts inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all Define staged construction parts are in

- **func** (function)

Function to call for each Define staged construction part

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the Define staged construction parts in model m:

```
StagedConstructionPart.ForEach(m, test);
function test(scp)
{
// scp is StagedConstructionPart object
}
```

To call function test for all of the Define staged construction parts in model m with optional object:

```
var data = { x:0, y:0 };
StagedConstructionPart.ForEach(m, test, data);
function test(scp, extra)
{
// scp is StagedConstructionPart object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of StagedConstructionPart objects or properties for all of the Define staged construction parts in a model in PRIMER. If the optional property argument is not given then an array of StagedConstructionPart objects is returned. If the property argument is given, that property value for each Define staged construction part is returned in the array instead of a StagedConstructionPart object

Arguments

- **Model** (Model)

Model to get Define staged construction parts from

- **property (optional)** (string)

Name for property to get for all Define staged construction parts in the model

Returns

Array of StagedConstructionPart objects or properties

Return type

Array

Example

To make an array of StagedConstructionPart objects for all of the Define staged construction parts in model m:

```
var a = StagedConstructionPart.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each Define staged construction part in model m:

```
var a = StagedConstructionPart.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a Define staged construction part.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the Define staged construction part scp:

```
var comm_array = scp.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of StagedConstructionPart objects for all of the flagged Define staged construction parts in a model in PRIMER If the optional property argument is not given then an array of StagedConstructionPart objects is returned. If the property argument is given, then that property value for each Define staged construction part is returned in the array instead of a StagedConstructionPart object

Arguments

- **Model** (Model)

Model to get Define staged construction parts from

- **flag** (Flag)

Flag set on the Define staged construction parts that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged Define staged construction parts in the model

Returns

Array of StagedConstructionPart objects or properties

Return type

Array

Example

To make an array of StagedConstructionPart objects for all of the Define staged construction parts in model m flagged with f:

```
var scp = StagedConstructionPart.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the Define staged construction parts in model m flagged with f:

```
var a = StagedConstructionPart.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the StagedConstructionPart object for a Define staged construction part ID.

Arguments

- **Model** (Model)

Model to find the Define staged construction part in

- **number** (integer)

number of the Define staged construction part you want the StagedConstructionPart object for

Returns

StagedConstructionPart object (or null if Define staged construction part does not exist).

Return type

StagedConstructionPart

Example

To get the StagedConstructionPart object for Define staged construction part 100 in model m

```
var scp = StagedConstructionPart.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a StagedConstructionPart property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the StagedConstructionPart.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

Define staged construction part property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if StagedConstructionPart property scp.example is a parameter:

```
Options.property_parameter_names = true;
if (scp.GetParameter(scp.example) ) do_something...
Options.property_parameter_names = false;
```

To check if StagedConstructionPart property scp.example is a parameter by using the GetParameter method:

```
if (scp.ViewParameters().GetParameter(scp.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this Define staged construction parts (*Define_staged_construction_part). **Note that a carriage return is not added**. See also StagedConstructionPart.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for Define staged construction parts scp:

```
var key = scp.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the Define staged construction parts. **Note that a carriage return is not added**. See also StagedConstructionPart.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for Define staged construction parts scp:

```
var cards = scp.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last Define staged construction part in the model.

Arguments

- **Model** (Model)

Model to get last Define staged construction part in

Returns

StagedConstructionPart object (or null if there are no Define staged construction parts in the model).

Return type

StagedConstructionPart

Example

To get the last Define staged construction part in model m:

```
var scp = StagedConstructionPart.Last(m);
```

Next()

Description

Returns the next Define staged construction part in the model.

Arguments

No arguments

Returns

StagedConstructionPart object (or null if there are no more Define staged construction parts in the model).

Return type

StagedConstructionPart

Example

To get the Define staged construction part in model m after Define staged construction part scp:

```
var scp = scp.Next();
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a Define staged construction part.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only Define staged construction parts from that model can be picked. If the argument is a Flag then only Define staged construction parts that are flagged with *limit* can be selected. If omitted, or null, any Define staged construction parts from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

StagedConstructionPart object (or null if not picked)

Return type

StagedConstructionPart

Example

To pick a Define staged construction part from model m giving the prompt 'Pick Define staged construction part from screen':

```
var scp = StagedConstructionPart.Pick('Pick Define staged construction part from screen', m);
```

Previous()

Description

Returns the previous Define staged construction part in the model.

Arguments

No arguments

Returns

StagedConstructionPart object (or null if there are no more Define staged construction parts in the model).

Return type

StagedConstructionPart

Example

To get the Define staged construction part in model m before Define staged construction part scp:

```
var scp = scp.Previous();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select Define staged construction parts using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting Define staged construction parts

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only Define staged construction parts from that model can be selected. If the argument is a Flag then only Define staged construction parts that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any Define staged construction parts can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of Define staged construction parts selected or null if menu cancelled

Return type

Number

Example

To select Define staged construction parts from model m, flagging those selected with flag f, giving the prompt 'Select Define staged construction parts':

```
StagedConstructionPart.Select(f, 'Select Define staged construction parts', m);
```

To select Define staged construction parts, flagging those selected with flag f but limiting selection to Define staged construction parts flagged with flag l, giving the prompt 'Select Define staged construction parts':

```
StagedConstructionPart.Select(f, 'Select Define staged construction parts', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the Define staged construction part.

Arguments

- **flag** (Flag)

Flag to set on the Define staged construction part

Returns

No return value

Example

To set flag f for Define staged construction part scp:

```
scp.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the Define staged construction part. The Define staged construction part will be sketched until you either call StagedConstructionPart.Unsketch(), StagedConstructionPart.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the Define staged construction part is sketched. If omitted redraw is true. If you want to sketch several Define staged construction parts and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch Define staged construction part scp:

```
scp.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged Define staged construction parts in the model. The Define staged construction parts will be sketched until you either call StagedConstructionPart.Unsketch(), StagedConstructionPart.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged Define staged construction parts will be sketched in

- **flag** (Flag)

Flag set on the Define staged construction parts that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the Define staged construction parts are sketched. If omitted redraw is true. If you want to sketch flagged Define staged construction parts several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all Define staged construction parts flagged with flag in model m:

```
StagedConstructionPart.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of Define staged construction parts in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing Define staged construction parts should be counted. If false or omitted referenced but undefined Define staged construction parts will also be included in the total.

Returns

number of Define staged construction parts

Return type

Number

Example

To get the total number of Define staged construction parts in model m:

```
var total = StagedConstructionPart.Total(m);
```

Unblank()

Description

Unblanks the Define staged construction part

Arguments

No arguments

Returns

No return value

Example

To unblank Define staged construction part scp:

```
scp.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the Define staged construction parts in the model.

Arguments

- **Model** (Model)

Model that all Define staged construction parts will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the Define staged construction parts in model m:

```
StagedConstructionPart.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged Define staged construction parts in the model.

Arguments

- **Model** (Model)

Model that the flagged Define staged construction parts will be unblanked in

- **flag** (Flag)

Flag set on the Define staged construction parts that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the Define staged construction parts in model m flagged with f:

```
StagedConstructionPart.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the Define staged construction parts in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all Define staged construction parts will be unset in

- **flag** (Flag)

Flag to unset on the Define staged construction parts

Returns

No return value

Example

To unset the flag f on all the Define staged construction parts in model m:

```
StagedConstructionPart.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the Define staged construction part.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the Define staged construction part is unsketched. If omitted redraw is true. If you want to unsketch several Define staged construction parts and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch Define staged construction part scp:

```
scp.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all Define staged construction parts.

Arguments

- **Model** (Model)

Model that all Define staged construction parts will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the Define staged construction parts are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all Define staged construction parts in model m:

```
StagedConstructionPart.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged Define staged construction parts in the model.

Arguments

- **Model** (Model)

Model that all Define staged construction parts will be unsketched in

- **flag** (Flag)

Flag set on the Define staged construction parts that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the Define staged construction parts are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all Define staged construction parts flagged with flag in model m:

```
StagedConstructionPart.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

StagedConstructionPart object.

Return type

StagedConstructionPart

Example

To check if StagedConstructionPart property scp.example is a parameter by using the StagedConstructionPart.GetParameter() method:

```
if (scp.ViewParameters().GetParameter(scp.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for Define staged construction part with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The Define staged construction part object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for Define staged construction part scp:

```
scp.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for Define staged construction part scp:

```
scp.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for Define staged construction part. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for Define staged construction part scp:

```
scp.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this Define staged construction part.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for Define staged construction part scp:

```
var xrefs = scp.Xrefs();
```

---

toString()

Description

Creates a string containing the Define staged construction parts data in keyword format. Note that this contains the keyword header and the keyword cards. See also StagedConstructionPart.Keyword() and StagedConstructionPart.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for Define staged construction parts scp in keyword format

```
var s = scp.toString();
```

Transformation class

The Transformation class gives you access to define transform cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)

Member functions

- AddRow(data*[Array of data]*, row (optional)*[integer]*)
- AssociateComment(Comment*[Comment]*)
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- GetRow(row*[integer]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- RemoveRow(row*[integer]*)
- SetFlag(flag*[Flag]*)
- SetRow(row*[integer]*, data*[Array of data]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

Transformation properties

Name	Type	Description
exists (read only)	logical	true if transformation exists, false if referred to but not defined.
include	integer	The Include file number that the transformation is in.

label	integer	Transformation number. Also see the tranid property which is an alternative name for this.
model (read only)	integer	The Model number that the transformation is in.
nrow (read only)	integer	Number of rows of transformations
title	string	The title for the transformation.
tranid	integer	Transformation number. Also see the label property which is an alternative name for this.

Detailed Description

The Transformation class allows you to create, modify, edit and manipulate define transformation cards. See the documentation below for more details.

Constructor

new Transformation(Model*[Model]*, tranid*[integer]*, title (optional)*[string]*)

Description

Create a new Transformation object.

Arguments

- **Model** (Model)

Model that transformation will be created in

- **tranid** (integer)

Transformation label

- **title (optional)** (string)

Transformation title

Returns

Transformation object

Return type

Transformation

Example

To create a new transformation in model m with label 1000 and title "Example transform"

```
var t = new Transformation(m, 1000, "Example transform");
```

Details of functions

AddRow(data*[Array of data]*, row (optional)*[integer]*)

Description

Adds a row of data for a *DEFINE_TRANSFORMATION.

Arguments

- **data** (Array of data)

The data you want to add

- **row (optional)** (integer)

The row you want to add the data at. Existing transforms will be shifted. If omitted the data will be added to the end of the existing transforms. **Note that row indices start at 0**.

Returns

No return value.

Example

To add a translation of (0, 0, 100) to transformation t:

```
var array = ["TRANSL", 0, 0, 100];
t.AddRow(array);
```

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a transformation.

Arguments

- **Comment** (Comment)

Comment that will be attached to the transformation

Returns

No return value

Example

To associate comment c to the transformation t:

```
t.AssociateComment(c);
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse transformation t:

```
t.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the transformation.

Arguments

- **flag** (Flag)

Flag to clear on the transformation

Returns

No return value

Example

To clear flag f for transformation t:

```
t.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the transformation. The target include of the copied transformation can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

Transformation object

Return type

Transformation

Example

To copy transformation t into transformation z:

```
var z = t.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a transformation

Arguments

- **Model** (Model)

Model that the transformation will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

Transformation object (or null if not made).

Return type

Transformation

Example

To start creating a transformation in model m:

```
var t = Transformation.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a transformation.

Arguments

- **Comment** (Comment)

Comment that will be detached from the transformation

Returns

No return value

Example

To detach comment c from the transformation t:

```
t.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit transformation t:

```
t.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for transformation with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description

autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The transformation object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for transformation t:

```
t.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for transformation t:

```
t.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for transformation. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for transformation t:

```
t.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first transformation in the model.

Arguments

- **Model** (Model)

Model to get first transformation in

Returns

Transformation object (or null if there are no transformations in the model).

Return type

Transformation

Example

To get the first transformation in model m:

```
var t = Transformation.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free transformation label in the model. Also see Transformation.LastFreeLabel(), Transformation.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free transformation label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

Transformation label.

Return type

Number

Example

To get the first free transformation label in model m:

```
var label = Transformation.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the transformations in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all transformations will be flagged in

- **flag** (Flag)

Flag to set on the transformations

Returns

No return value

Example

To flag all of the transformations with flag f in model m:

```
Transformation.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the transformation is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the transformation

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if transformation t has flag f set on it:

```
if (t.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each transformation in the model.
Note that ForEach has been designed to make looping over transformations as fast as possible and so has some limitations.
Firstly, a single temporary Transformation object is created and on each function call it is updated with the current transformation data. This means that you should not try to store the Transformation object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new transformations inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all transformations are in

- **func** (function)

Function to call for each transformation

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the transformations in model m:

```
Transformation.ForEach(m, test);
function test(t)
{
// t is Transformation object
}
```

To call function test for all of the transformations in model m with optional object:

```
var data = { x:0, y:0 };
Transformation.ForEach(m, test, data);
function test(t, extra)
{
// t is Transformation object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Transformation objects or properties for all of the transformations in a model in PRIMER. If the optional property argument is not given then an array of Transformation objects is returned. If the property argument is given, that property value for each transformation is returned in the array instead of a Transformation object

Arguments

- **Model** (Model)

Model to get transformations from

- **property (optional)** (string)

Name for property to get for all transformations in the model

Returns

Array of Transformation objects or properties

Return type

Array

Example

To make an array of Transformation objects for all of the transformations in model m:

```
var a = Transformation.GetAll(m);
```

To return an array containing the value of property ’foo’ (for example ’x’ for a node) for each transformation in model m:

```
var a = Transformation.GetAll(m, ’foo’);
```

GetComments()

Description

Extracts the comments associated to a transformation.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the transformation t:

```
var comm_array = t.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of Transformation objects for all of the flagged transformations in a model in PRIMER If the optional property argument is not given then an array of Transformation objects is returned. If the property argument is given, then that property value for each transformation is returned in the array instead of a Transformation object

Arguments

- **Model** (Model)

Model to get transformations from

- **flag** (Flag)

Flag set on the transformations that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged transformations in the model

Returns

Array of Transformation objects or properties

Return type

Array

Example

To make an array of Transformation objects for all of the transformations in model m flagged with f:

```
var t = Transformation.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the transformations in model m flagged with f:

```
var a = Transformation.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the Transformation object for a transformation ID.

Arguments

- **Model** (Model)

Model to find the transformation in

- **number** (integer)

number of the transformation you want the Transformation object for

Returns

Transformation object (or null if transformation does not exist).

Return type

Transformation

Example

To get the Transformation object for transformation 100 in model m

```
var t = Transformation.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a Transformation property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the Transformation.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

transformation property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if Transformation property t.example is a parameter:

```
Options.property_parameter_names = true;
if (t.GetParameter(t.example) ) do_something...
Options.property_parameter_names = false;
```

To check if Transformation property t.example is a parameter by using the GetParameter method:

```
if (t.ViewParameters().GetParameter(t.example) ) do_something...
```

GetRow(row*[integer]*)

Description

Returns the data for a row in the transformation.

Arguments

- **row** (integer)

The row you want the data for. **Note row indices start at 0**.

Returns

An array of numbers containing the row variables.

Return type

Number

Example

To get the data for the 2nd row in transformation t:

```
var data = t.GetRow(1);
```

Keyword()

Description

Returns the keyword for this transformation. **Note that a carriage return is not added**. See also Transformation.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for transformation t:

```
var key = t.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the transformation. **Note that a carriage return is not added**. See also Transformation.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for transformation i:

```
var cards = i.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last transformation in the model.

Arguments

- **Model** (Model)

Model to get last transformation in

Returns

Transformation object (or null if there are no transformations in the model).

Return type

Transformation

Example

To get the last transformation in model m:

```
var t = Transformation.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free transformation label in the model. Also see Transformation.FirstFreeLabel(), Transformation.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free transformation label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

Transformation label.

Return type

Number

Example

To get the last free transformation label in model m:

```
var label = Transformation.LastFreeLabel(m);
```

Next()

Description

Returns the next transformation in the model.

Arguments

No arguments

Returns

Transformation object (or null if there are no more transformations in the model).

Return type

Transformation

Example

To get the transformation in model m after transformation t:

```
var t = t.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) transformation label in the model. Also see Transformation.FirstFreeLabel(), Transformation.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free transformation label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

Transformation label.

Return type

Number

Example

To get the next free transformation label in model m:

```
var label = Transformation.NextFreeLabel(m);
```

Previous()

Description

Returns the previous transformation in the model.

Arguments

No arguments

Returns

Transformation object (or null if there are no more transformations in the model).

Return type

Transformation

Example

To get the transformation in model m before transformation t:

```
var t = t.Previous();
```

RemoveRow(row*[integer]*)

Description

Removes the data for a row in *DEFINE_TRANSFORMATION.

Arguments

- **row** (integer)

The row you want to remove the data for. **Note that row indices start at 0**.

Returns

No return value.

Example

To remove the second row of data for transformation t:

```
t.RemoveRow(1);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select transformations using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting transformations

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only transformations from that model can be selected. If the argument is a Flag then only transformations that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any transformations can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of transformations selected or null if menu cancelled

Return type

Number

Example

To select transformations from model m, flagging those selected with flag f, giving the prompt 'Select transformations':

```
Transformation.Select(f, 'Select transformations', m);
```

To select transformations, flagging those selected with flag f but limiting selection to transformations flagged with flag l, giving the prompt 'Select transformations':

```
Transformation.Select(f, 'Select transformations', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the transformation.

Arguments

- **flag** (Flag)

Flag to set on the transformation

Returns

No return value

Example

To set flag f for transformation t:

```
t.SetFlag(f);
```

SetRow(row*[integer]*, data*[Array of data]*)

Description

Sets the data for a row in *DEFINE_TRANSFORMATION.

Arguments

- **row** (integer)

The row you want to set the data for. **Note that row indices start at 0**.

- **data** (Array of data)

The data you want to set the row to

Returns

No return value.

Example

To set the second row of data for transformation t to be a translation of (0, 0, 100):

```
var array = ["TRANSL", 0, 0, 100];
t.SetRow(1, array);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of transformations in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing transformations should be counted. If false or omitted referenced but undefined transformations will also be included in the total.

Returns

number of transformations

Return type

Number

Example

To get the total number of transformations in model m:

```
var total = Transformation.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the transformations in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all transformations will be unset in

- **flag** (Flag)

Flag to unset on the transformations

Returns

No return value

Example

To unset the flag f on all the transformations in model m:

```
Transformation.UnflagAll(m, f);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

Transformation object.

Return type

Transformation

Example

To check if Transformation property t.example is a parameter by using the Transformation.GetParameter() method:

```
if (t.ViewParameters().GetParameter(t.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for transformation with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The transformation object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for transformation t:

```
t.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for transformation t:

```
t.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for transformation. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for transformation t:

```
t.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this transformation.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for transformation t:

```
var xrefs = t.Xrefs();
```

---

toString()

Description

Creates a string containing the transformation data in keyword format. Note that this contains the keyword header and the keyword cards. See also Transformation.Keyword() and Transformation.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for transformation t in keyword format

```
var s = t.toString();
```

Vector class

The Vector class gives you access to define vector cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]

- Xrefs()
- toString()

Vector properties

Name	Type	Description
cid	int	Coordinate system ID
exists (read only)	logical	true if vector exists, false if referred to but not defined.
heading	string	Vector heading
include	integer	The Include file number that the vector is in.
label	integer	Vector number. Also see the vid property which is an alternative name for this.
model (read only)	integer	The Model number that the vector is in.
nodeh	int	Node ID for head of vector (for _NODES option)
nodes	logical	_NODES option
nodet	int	Node ID for tail of vector (for _NODES option)
vid	integer	Vector number. Also see the label property which is an alternative name for this.
xh	real	X coordinate of head of vector
xt	real	X coordinate of tail of vector
yh	real	Y coordinate of head of vector
yt	real	Y coordinate of tail of vector
zh	real	Z coordinate of head of vector
zt	real	Z coordinate of tail vector

Detailed Description

The Vector class allows you to create, modify, edit and manipulate vector cards. See the documentation below for more details.

Constructor

new Vector(Model*[Model]*, options*[object]*)

Description

Create a new Vector object.

Arguments

- **Model** (Model)

Model that vector will be created in

- **options** (object)

Options for creating the Vector

Object has the following properties:

Name	Type	Description
ch (optional)	Array of numbers	Array of coordinates of head of vector [xh, yh, zh] (for *DEFINE_VECTOR)
cid (optional)	int	Optional coordinate system ID (for *DEFINE_VECTOR)

ct (optional)	Array of numbers	Array of coordinates of tail of vector [xt, yt, zt] (for *DEFINE_VECTOR)
heading (optional)	string	Optional title for the vector
nodeh (optional)	integer	Node ID for head of vector (for *DEFINE_VECTOR_NODES)
nodes	boolean	_NODES option (true for *DEFINE_VECTOR_NODES, false for *DEFINE_VECTOR)
nodet (optional)	integer	Node ID for tail of vector (for *DEFINE_VECTOR_NODES)
vid	integer	Vector ID.

Returns

Vector object

Return type

Vector

Example

To create a new *DEFINE_VECTOR in model m with label 100 with the tail at (1.5, 2.5, 1.0) and the head at (4.5, 4.0, 3.0)

```
var v = new Vector(m, {nodes: 0, vid: 100, ct: [1.5, 2.5, 1.0], ch: [4.5, 4.0, 3.0]});
```

To create a new *DEFINE_VECTOR_NODES in model m with label 200 using node 10 for the tail and node 20 for the head

```
var v = new Vector(m, {nodes: 1, vid: 200, nodet: 10, nodeh: 20});
```

new Vector(Model*[Model]*, vid*[integer]*, xt*[real]*, yt*[real]*, zt*[real]*, xh*[real]*, yh*[real]*, zh*[real]*, cid (optional)*[int]*, heading (optional)*[string]*) [deprecated]

This function is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Create a new Vector object.

Arguments

- **Model** (Model)

Model that vector will be created in

- **vid** (integer)

Vector number

- **xt** (real)

X coordinate of tail of vector

- **yt** (real)

Y coordinate of tail of vector

- **zt** (real)

Z coordinate of tail vector

- **xh** (real)

X coordinate of head of vector

- **yh** (real)

Y coordinate of head of vector

- **zh** (real)

Z coordinate of head of vector

- **cid (optional)** (int)

Coordinate system ID

- **heading (optional)** (string)

Title for the vector

Returns

Vector object

Return type

Vector

Example

To create a new vector in model m with label 200

```
var v = new Vector(m, 200, 1.5, 2.5, 1.0, 4.5, 4.0, 3.0);
```

new Vector(Model*[Model]*, vid*[integer]*, nodet*[integer]*, nodeh*[integer]*, heading (optional)*[string]*) [deprecated]

This function is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Create a new Vector object with _NODES option.

Arguments

- **Model** (Model)

Model that vector will be created in

- **vid** (integer)

Vector number

- **nodet** (integer)

Node ID for tail of vector

- **nodeh** (integer)

Node ID for head of vector

- **heading (optional)** (string)

Title for the vector

Returns

Vector object

Return type

Vector

Example

To create a new vector in model m with label 200 using nodes 10 for the tail and 20 for the head

```
var v = new Vector(m, 200, 20, 30);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a vector.

Arguments

- **Comment** (Comment)

Comment that will be attached to the vector

Returns

No return value

Example

To associate comment c to the vector v:

```
v.AssociateComment(c);
```

Blank()

Description

Blanks the vector

Arguments

No arguments

Returns

No return value

Example

To blank vector v:

```
v.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the vectors in the model.

Arguments

- **Model** (Model)

Model that all vectors will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the vectors in model m:

```
Vector.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged vectors in the model.

Arguments

- **Model** (Model)

Model that all the flagged vectors will be blanked in

- **flag** (Flag)

Flag set on the vectors that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the vectors in model m flagged with f:

```
Vector.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the vector is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if vector v is blanked:

```
if (v.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse vector v:

```
v.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the vector.

Arguments

- **flag** (Flag)

Flag to clear on the vector

Returns

No return value

Example

To clear flag f for vector v:

```
v.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the vector. The target include of the copied vector can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

Vector object

Return type

Vector

Example

To copy vector v into vector z:

```
var z = v.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a vector

Arguments

- **Model** (Model)

Model that the vector will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

Vector object (or null if not made).

Return type

Vector

Example

To start creating a vector in model m:

```
var v = Vector.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a vector.

Arguments

- **Comment** (Comment)

Comment that will be detached from the vector

Returns

No return value

Example

To detach comment c from the vector v:

```
v.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit vector v:

```
v.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for vector with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The vector object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for vector v:

```
v.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for vector v:

```
v.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for vector. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for vector v:

```
v.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first vector in the model.

Arguments

- **Model** (Model)

Model to get first vector in

Returns

Vector object (or null if there are no vectors in the model).

Return type

Vector

Example

To get the first vector in model m:

```
var v = Vector.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free vector label in the model. Also see Vector.LastFreeLabel(), Vector.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free vector label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

Vector label.

Return type

Number

Example

To get the first free vector label in model m:

```
var label = Vector.FirstFreeLabel(m);
```

FlagAll(Model[*Model*], flag[*Flag*]) [static]

Description

Flags all of the vectors in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all vectors will be flagged in

- **flag** (Flag)

Flag to set on the vectors

Returns

No return value

Example

To flag all of the vectors with flag f in model m:

```
Vector.FlagAll(m, f);
```

Flagged(flag[*Flag*])

Description

Checks if the vector is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the vector

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if vector v has flag f set on it:

```
if (v.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each vector in the model.
Note that ForEach has been designed to make looping over vectors as fast as possible and so has some limitations.
Firstly, a single temporary Vector object is created and on each function call it is updated with the current vector data. This means that you should not try to store the Vector object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new vectors inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all vectors are in

- **func** (function)

Function to call for each vector

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the vectors in model m:

```
Vector.ForEach(m, test);
function test(v)
{
// v is Vector object
}
```

To call function test for all of the vectors in model m with optional object:

```
var data = { x:0, y:0 };
Vector.ForEach(m, test, data);
function test(v, extra)
{
// v is Vector object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Vector objects or properties for all of the vectors in a model in PRIMER. If the optional property argument is not given then an array of Vector objects is returned. If the property argument is given, that property value for each vector is returned in the array instead of a Vector object

Arguments

- **Model** (Model)

Model to get vectors from

- **property (optional)** (string)

Name for property to get for all vectors in the model

Returns

Array of Vector objects or properties

Return type

Array

Example

To make an array of Vector objects for all of the vectors in model m:

```
var a = Vector.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each vector in model m:

```
var a = Vector.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a vector.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the vector v:

```
var comm_array = v.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of Vector objects for all of the flagged vectors in a model in PRIMER If the optional property argument is not given then an array of Vector objects is returned. If the property argument is given, then that property value for each vector is returned in the array instead of a Vector object

Arguments

- **Model** (Model)

Model to get vectors from

- **flag** (Flag)

Flag set on the vectors that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged vectors in the model

Returns

Array of Vector objects or properties

Return type

Array

Example

To make an array of Vector objects for all of the vectors in model m flagged with f:

```
var v = Vector.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the vectors in model m flagged with f:

```
var a = Vector.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the Vector object for a vector ID.

Arguments

- **Model** (Model)

Model to find the vector in

- **number** (integer)

number of the vector you want the Vector object for

Returns

Vector object (or null if vector does not exist).

Return type

Vector

Example

To get the Vector object for vector 100 in model m

```
var v = Vector.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a Vector property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the Vector.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

vector property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if Vector property v.example is a parameter:

```
Options.property_parameter_names = true;
if (v.GetParameter(v.example) ) do_something...
Options.property_parameter_names = false;
```

To check if Vector property v.example is a parameter by using the GetParameter method:

```
if (v.ViewParameters().GetParameter(v.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this vector (*DEFINE_VECTOR). **Note that a carriage return is not added**. See also Vector.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for vector m:

```
var key = m.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the vector. **Note that a carriage return is not added**. See also Vector.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for vector v:

```
var cards = v.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last vector in the model.

Arguments

- **Model** (Model)

Model to get last vector in

Returns

Vector object (or null if there are no vectors in the model).

Return type

Vector

Example

To get the last vector in model m:

```
var v = Vector.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free vector label in the model. Also see Vector.FirstFreeLabel(), Vector.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free vector label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

Vector label.

Return type

Number

Example

To get the last free vector label in model m:

```
var label = Vector.LastFreeLabel(m);
```

Next()

Description

Returns the next vector in the model.

Arguments

No arguments

Returns

Vector object (or null if there are no more vectors in the model).

Return type

Vector

Example

To get the vector in model m after vector v:

```
var v = v.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) vector label in the model. Also see Vector.FirstFreeLabel(), Vector.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free vector label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

Vector label.

Return type

Number

Example

To get the next free vector label in model m:

```
var label = Vector.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a vector.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only vectors from that model can be picked. If the argument is a Flag then only vectors that are flagged with *limit* can be selected. If omitted, or null, any vectors from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

Vector object (or null if not picked)

Return type

Vector

Example

To pick a vector from model m giving the prompt 'Pick vector from screen':

```
var v = Vector.Pick('Pick vector from screen', m);
```

Previous()

Description

Returns the previous vector in the model.

Arguments

No arguments

Returns

Vector object (or null if there are no more vectors in the model).

Return type

Vector

Example

To get the vector in model m before vector v:

```
var v = v.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the vectors in the model.

Arguments

- **Model** (Model)

Model that all vectors will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the vectors in model m, from 1000000:

```
Vector.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged vectors in the model.

Arguments

- **Model** (Model)

Model that all the flagged vectors will be renumbered in

- **flag** (Flag)

Flag set on the vectors that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the vectors in model m flagged with f, from 1000000:

```
Vector.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select vectors using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting vectors

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only vectors from that model can be selected. If the argument is a Flag then only vectors that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any vectors can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of vectors selected or null if menu cancelled

Return type

Number

Example

To select vectors from model m, flagging those selected with flag f, giving the prompt 'Select vectors':

```
Vector.Select(f, 'Select vectors', m);
```

To select vectors, flagging those selected with flag f but limiting selection to vectors flagged with flag l, giving the prompt 'Select vectors':

```
Vector.Select(f, 'Select vectors', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the vector.

Arguments

- **flag** (Flag)

Flag to set on the vector

Returns

No return value

Example

To set flag f for vector v:

```
v.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the vector. The vector will be sketched until you either call Vector.Unsketch(), Vector.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the vector is sketched. If omitted redraw is true. If you want to sketch several vectors and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch vector v:

```
v.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged vectors in the model. The vectors will be sketched until you either call Vector.Unsketch(), Vector.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged vectors will be sketched in

- **flag** (Flag)

Flag set on the vectors that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the vectors are sketched. If omitted redraw is true. If you want to sketch flagged vectors several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all vectors flagged with flag in model m:

```
Vector.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of vectors in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing vectors should be counted. If false or omitted referenced but undefined vectors will also be included in the total.

Returns

number of vectors

Return type

Number

Example

To get the total number of vectors in model m:

```
var total = Vector.Total(m);
```

Unblank()

Description

Unblanks the vector

Arguments

No arguments

Returns

No return value

Example

To unblank vector v:

```
v.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the vectors in the model.

Arguments

- **Model** (Model)

Model that all vectors will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the vectors in model m:

```
Vector.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged vectors in the model.

Arguments

- **Model** (Model)

Model that the flagged vectors will be unblanked in

- **flag** (Flag)

Flag set on the vectors that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the vectors in model m flagged with f:

```
Vector.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the vectors in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all vectors will be unset in

- **flag** (Flag)

Flag to unset on the vectors

Returns

No return value

Example

To unset the flag f on all the vectors in model m:

```
Vector.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the vector.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the vector is unsketched. If omitted redraw is true. If you want to unsketch several vectors and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch vector v:

```
v.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all vectors.

Arguments

- **Model** (Model)

Model that all vectors will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the vectors are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all vectors in model m:

```
Vector.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged vectors in the model.

Arguments

- **Model** (Model)

Model that all vectors will be unsketched in

- **flag** (Flag)

Flag set on the vectors that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the vectors are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all vectors flagged with flag in model m:

```
Vector.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

Vector object.

Return type

Vector

Example

To check if Vector property v.example is a parameter by using the Vector.GetParameter() method:

```
if (v.ViewParameters().GetParameter(v.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for vector with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The vector object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for vector v:

```
v.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for vector v:

```
v.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for vector. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for vector v:

```
v.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this vector.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for vector v:

```
var xrefs = v.Xrefs();
```

---

toString()

Description

Creates a string containing the vector data in keyword format. Note that this contains the keyword header and the keyword cards. See also Vector.Keyword() and Vector.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for vector v in keyword format

```
var s = v.toString();
```

DeformableToRigid class

The DeformableToRigid class gives you access to *DEFORMABLE_TO_RIGID cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model *[Model]*, redraw (optional) *[boolean]*)
- BlankFlagged(Model *[Model]*, flag *[Flag]*, redraw (optional) *[boolean]*)
- Create(Model *[Model]*, modal (optional) *[boolean]*)
- First(Model *[Model]*)
- FirstFreeLabel(Model *[Model]*, layer (optional) *[Include number]*)
- FlagAll(Model *[Model]*, flag *[Flag]*)
- ForEach(Model *[Model]*, func *[function]*, extra (optional) *[any]*)
- GetAll(Model *[Model]*, property (optional) *[string]*)
- GetFlagged(Model *[Model]*, flag *[Flag]*, property (optional) *[string]*)
- GetFromID(Model *[Model]*, number *[integer]*)
- Last(Model *[Model]*)
- LastFreeLabel(Model *[Model]*, layer (optional) *[Include number]*)
- NextFreeLabel(Model *[Model]*, layer (optional) *[Include number]*)
- Pick(prompt *[string]*, limit (optional) *[Model or Flag]*, modal (optional) *[boolean]*, button text (optional) *[string]*)
- RenumberAll(Model *[Model]*, start *[integer]*)
- RenumberFlagged(Model *[Model]*, flag *[Flag]*, start *[integer]*)
- Select(flag *[Flag]*, prompt *[string]*, limit (optional) *[Model or Flag]*, modal (optional) *[boolean]*)
- SketchFlagged(Model *[Model]*, flag *[Flag]*, redraw (optional) *[boolean]*)
- Total(Model *[Model]*, exists (optional) *[boolean]*)
- UnblankAll(Model *[Model]*, redraw (optional) *[boolean]*)
- UnblankFlagged(Model *[Model]*, flag *[Flag]*, redraw (optional) *[boolean]*)
- UnflagAll(Model *[Model]*, flag *[Flag]*)
- UnsketchAll(Model *[Model]*, redraw (optional) *[boolean]*)
- UnsketchFlagged(Model *[Model]*, flag *[Flag]*, redraw (optional) *[boolean]*)

Member functions

- AssociateComment(Comment *[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional) *[boolean]*)
- ClearFlag(flag *[Flag]*)
- Copy(range (optional) *[boolean]*)
- DetachComment(Comment *[Comment]*)
- Edit(modal (optional) *[boolean]*)
- Error(message *[string]*, options (optional) *[object]*)
- Error(message *[string]*, details (optional) *[string]*) [deprecated]
- Flagged(flag *[Flag]*)
- GetComments()
- GetDefToRegAutoCard(ctype *[integer]*, index *[integer]*)
- GetParameter(prop *[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- RemoveDefToRegAutoCard(ctype *[integer]*, index *[integer]*)
- SetDefToRegAutoCard(ctype *[integer]*, index *[integer]*, ptype *[integer]*, pid *[integer]*, lrb (optional) *[integer]*)
- SetFlag(flag *[Flag]*)
- Sketch(redraw (optional) *[boolean]*)
- Unblank()
- Unsketch(redraw (optional) *[boolean]*)

- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

DeformableToRigid constants

Constants for DEFORMABLE_TO_RIGID type

Name	Description
DeformableToRigid.AUTOMATIC	*DEFORMABLE_TO_RIGID_AUTOMATIC.
DeformableToRigid.INERTIA	*DEFORMABLE_TO_RIGID_INERTIA.
DeformableToRigid.SIMPLE	*DEFORMABLE_TO_RIGID.

Constants for PID field type

Name	Description
DeformableToRigid.PART	Identifies the PID type as Part. Used for field ptype. Used only for DeformableToRigid.SIMPLE or DeformableToRigid.INERTIA.
DeformableToRigid.PSET	Identifies the PID type as Part Set. Used for field ptype. Used only for DeformableToRigid.SIMPLE or DeformableToRigid.INERTIA.

Constants for automatic types

Name	Description
DeformableToRigid.D2R	Identifies that card is being written/retreived/removed as D2R card. Used in methods GetDefToRegAutoCard, SetDefToRegAutoCard and RemoveDefToRegAutoCard. Used only for DeformableToRigid.AUTOMATIC.
DeformableToRigid.R2D	Identifies that card is being written/retreived/removed as R2D card. Used in methods GetDefToRegAutoCard, SetDefToRegAutoCard and RemoveDefToRegAutoCard. Used only for DeformableToRigid.AUTOMATIC.

DeformableToRigid properties

Name	Type	Description
code	integer	Activation switch code. (Valid values: 0-5). Used only for DeformableToRigid.AUTOMATIC.
d2r	integer	Number of deformable parts to be switched to rigid plus number of rigid parts for which new lead/constrained rigid body combinations will be defined. Used only for DeformableToRigid.AUTOMATIC.
dtmax	real	Maximum permitted time step size after switch. Used only for DeformableToRigid.AUTOMATIC.
entno	integer	Rigid wall/contact surface number for switch codes 1, 2, 3, 4. Used only for DeformableToRigid.AUTOMATIC.
exists (read only)	logical	true if deformable to rigid exists, false if referred to but not defined.
include	integer	The Include file number that the deformable to rigid is in.
ixx	real	The xx component of inertia tensor. Used only for DeformableToRigid.INERTIA.
ixx	real	The xx component of inertia tensor. Used only for DeformableToRigid.INERTIA.

ixy	real	The xy component of inertia tensor. Used only for DeformableToRigid.INERTIA.
ixz	real	The xz component of inertia tensor. Used only for DeformableToRigid.INERTIA.
iyz	real	The yz component of inertia tensor. Used only for DeformableToRigid.INERTIA.
izz	real	The zz component of inertia tensor. Used only for DeformableToRigid.INERTIA.
lrb	integer	Part ID of the lead rigid body to which the part is merged. Used only for DeformableToRigid.SIMPLE.
model (read only)	integer	The Model number that the deformable to rigid is in.
ncsf	integer	Nodal constraint body flag. (Valid values : 0, 1, 2). Used only for DeformableToRigid.AUTOMATIC.
nrbf	integer	Nodal rigid body flag. (Valid values : 0, 1, 2). Used only for DeformableToRigid.AUTOMATIC.
offset	real	Optional contact thickness for switch to deformable. Used only for DeformableToRigid.AUTOMATIC.
paired	integer	Define a pair of related switches. (Valid values : -1, 0, 1). Used only for DeformableToRigid.AUTOMATIC.
pid	integer	Part or Part set ID which is switched to a rigid material. Depends on value of ptype. Used only for DeformableToRigid.SIMPLE or DeformableToRigid.INERTIA.
ptype	integer	Type of PID. Valid values are: DeformableToRigid.PART or DeformableToRigid.PSET. Used only for DeformableToRigid.SIMPLE.
r2d	integer	Number of rigid parts to be switched to deformable. Used only for DeformableToRigid.AUTOMATIC.
relsw	integer	Related switch set. Used only for DeformableToRigid.AUTOMATIC.
rwf	integer	Flag to delete or activate rigid walls. (Valid values : 0, 1, 2). Used only for DeformableToRigid.AUTOMATIC.
swset (read only)	integer	Set number for this automatic switch set. Used only for DeformableToRigid.AUTOMATIC.
time1	real	Switch will not take place before this time. Used only for DeformableToRigid.AUTOMATIC.
time2	real	Switch will not take place after this time. Used only for DeformableToRigid.AUTOMATIC.
time3	real	After this part switch has taken place, another automatic switch will not take place for the duration of the delay period. Used only for DeformableToRigid.AUTOMATIC.
tm	real	Translational mass. Used only for DeformableToRigid.INERTIA.
type (read only)	integer	Gives the type of DeformableToRigid Object.
xc	real	x-coordinate of center of mass. Used only for DeformableToRigid.INERTIA.
yc	real	y-coordinate of center of mass. Used only for DeformableToRigid.INERTIA.
zc	real	z-coordinate of center of mass. Used only for DeformableToRigid.INERTIA.

Detailed Description

The DeformableToRigid class allows you to create, modify, edit and manipulate deformable to rigid cards. See the documentation below for more details.

Constructor

new DeformableToRigid(Model*[Model]*, Type*[constant]*, pid (optional) *[integer]*, lrb (optional)*[integer]*, ptype (optional)*[integer]*)

Description

Create a new DeformableToRigid object.

Arguments

- **Model** (Model)

Model that deformable to rigid will be created in

- **Type** (constant)

Specify the type of DeformableToRigid (Can be DeformableToRigid.SIMPLE or DeformableToRigid.AUTOMATIC or DeformableToRigid.INERTIA)

- **pid (optional)** (integer)

Part or Part set ID which is switched to a rigid material. Depends on value of ptype. Used only for DeformableToRigid.SIMPLE or DeformableToRigid.INERTIA.

- **lrb (optional)** (integer)

Part ID of the lead rigid body to which the part is merged. Used only for DeformableToRigid.SIMPLE.

- **ptype (optional)** (integer)

Type of PID. Valid values are: DeformableToRigid.PART or DeformableToRigid.PSET. Used only for DeformableToRigid.SIMPLE.

Returns

DeformableToRigid object

Return type

DeformableToRigid

Example

To create a new deformable to rigid in model m, type SIMPLE, part id 100:

```
var dtor = new DeformableToRigid(m, DeformableToRigid.SIMPLE, 100);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a deformable to rigid.

Arguments

- **Comment** (Comment)

Comment that will be attached to the deformable to rigid

Returns

No return value

Example

To associate comment c to the deformable to rigid dtor:

```
dtor.AssociateComment(c);
```

Blank()

Description

Blanks the deformable to rigid

Arguments

No arguments

Returns

No return value

Example

To blank deformable to rigid dtor:

```
dtor.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the deformable to rigids in the model.

Arguments

- **Model** (Model)

Model that all deformable to rigids will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the deformable to rigids in model m:

```
DeformableToRigid.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged deformable to rigids in the model.

Arguments

- **Model** (Model)

Model that all the flagged deformable to rigids will be blanked in

- **flag** (Flag)

Flag set on the deformable to rigids that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the deformable to rigids in model m flagged with f:

```
DeformableToRigid.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the deformable to rigid is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if deformable to rigid dtor is blanked:

```
if (dtor.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse deformable to rigid dtor:

```
dtor.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the deformable to rigid.

Arguments

- **flag** (Flag)

Flag to clear on the deformable to rigid

Returns

No return value

Example

To clear flag f for deformable to rigid dtor:

```
dtor.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the deformable to rigid. The target include of the copied deformable to rigid can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

DeformableToRigid object

Return type

DeformableToRigid

Example

To copy deformable to rigid dtor into deformable to rigid z:

```
var z = dtor.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a deformable to rigid

Arguments

- **Model** (Model)

Model that the deformable to rigid will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

DeformableToRigid object (or null if not made).

Return type

DeformableToRigid

Example

To start creating a deformable to rigid in model m:

```
var dtor = DeformableToRigid.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a deformable to rigid.

Arguments

- **Comment** (Comment)

Comment that will be detached from the deformable to rigid

Returns

No return value

Example

To detach comment c from the deformable to rigid dtor:

```
dtor.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit deformable to rigid dtor:

```
dtor.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for deformable to rigid with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The deformable to rigid object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for deformable to rigid dtor:

```
dtor.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for deformable to rigid dtor:

```
dtor.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for deformable to rigid. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for deformable to rigid dtor:

```
dtor.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first deformable to rigid in the model.

Arguments

- **Model** (Model)

Model to get first deformable to rigid in

Returns

DeformableToRigid object (or null if there are no deformable to rigids in the model).

Return type

DeformableToRigid

Example

To get the first deformable to rigid in model m:

```
var dtor = DeformableToRigid.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free deformable to rigid label in the model. Also see DeformableToRigid.LastFreeLabel(), DeformableToRigid.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free deformable to rigid label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

DeformableToRigid label.

Return type

Number

Example

To get the first free deformable to rigid label in model m:

```
var label = DeformableToRigid.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the deformable to rigids in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all deformable to rigids will be flagged in

- **flag** (Flag)

Flag to set on the deformable to rigids

Returns

No return value

Example

To flag all of the deformable to rigids with flag f in model m:

```
DeformableToRigid.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the deformable to rigid is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the deformable to rigid

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if deformable to rigid dtor has flag f set on it:

```
if (dtor.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each deformable to rigid in the model.
Note that ForEach has been designed to make looping over deformable to rigids as fast as possible and so has some limitations.
Firstly, a single temporary DeformableToRigid object is created and on each function call it is updated with the current deformable to rigid data. This means that you should not try to store the DeformableToRigid object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new deformable to rigids inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all deformable to rigids are in

- **func** (function)

Function to call for each deformable to rigid

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the deformable to rigids in model m:

```
DeformableToRigid.ForEach(m, test);
function test(dtor)
{
// dtor is DeformableToRigid object
}
```

To call function test for all of the deformable to rigids in model m with optional object:

```
var data = { x:0, y:0 };
DeformableToRigid.ForEach(m, test, data);
function test(dtor, extra)
{
// dtor is DeformableToRigid object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of DeformableToRigid objects or properties for all of the deformable to rigids in a model in PRIMER. If the optional property argument is not given then an array of DeformableToRigid objects is returned. If the property argument is given, that property value for each deformable to rigid is returned in the array instead of a DeformableToRigid object

Arguments

- **Model** (Model)

Model to get deformable to rigids from

- **property (optional)** (string)

Name for property to get for all deformable to rigids in the model

Returns

Array of DeformableToRigid objects or properties

Return type

Array

Example

To make an array of DeformableToRigid objects for all of the deformable to rigids in model m:

```
var a = DeformableToRigid.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each deformable to rigid in model m:

```
var a = DeformableToRigid.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a deformable to rigid.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the deformable to rigid dtor:

```
var comm_array = dtor.GetComments();
```

GetDefToRegAutoCard(ctype*[integer]*, index*[integer]*)

Description

Returns the D2R or R2D cards for *DEFORMABLE_TO_RIGID_AUTOMATC.

Arguments

- **ctype** (integer)

The card type you want the data for. Can be D2R or R2D.

- **index** (integer)

The card index you want the data for. **Note that card indices start at 0, not 1**.

Returns

An array of numbers containing the 2 or 3 member (depending on Card type): Part or Part Set ID, LRB Part ID (only for card type D2R), and part type (PTYPE - Can be DeformableToRigid.PART or DeformableToRigid.PSET).

Return type

Number

Example

To get the D2R card data for the 3rd D2R card for Deformable to Rigid dtor:

```
if (dtor.d2r >= 3)
{
    var dtor_data = dtor.GetDefToRegAutoCard(DeformableToRigid.D2R, 2);
}
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of DeformableToRigid objects for all of the flagged deformable to rigids in a model in PRIMER If the optional property argument is not given then an array of DeformableToRigid objects is returned. If the property argument is given, then that property value for each deformable to rigid is returned in the array instead of a DeformableToRigid object

Arguments

- **Model** (Model)

Model to get deformable to rigids from

- **flag** (Flag)

Flag set on the deformable to rigids that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged deformable to rigids in the model

Returns

Array of DeformableToRigid objects or properties

Return type

Array

Example

To make an array of DeformableToRigid objects for all of the deformable to rigids in model m flagged with f:

```
var dtor = DeformableToRigid.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the deformable to rigids in model m flagged with f:

```
var a = DeformableToRigid.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the DeformableToRigid object for a deformable to rigid ID.

Arguments

- **Model** (Model)

Model to find the deformable to rigid in

- **number** (integer)

number of the deformable to rigid you want the DeformableToRigid object for

Returns

DeformableToRigid object (or null if deformable to rigid does not exist).

Return type

DeformableToRigid

Example

To get the DeformableToRigid object for deformable to rigid 100 in model m

```
var dtor = DeformableToRigid.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a DeformableToRigid property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the DeformableToRigid.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

deformable to rigid property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if DeformableToRigid property dtor.example is a parameter:

```
Options.property_parameter_names = true;
if (dtor.GetParameter(dtor.example) ) do_something...
Options.property_parameter_names = false;
```

To check if DeformableToRigid property dtor.example is a parameter by using the GetParameter method:

```
if (dtor.ViewParameters().GetParameter(dtor.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this DeformableToRigid (*DEFORMABLE_TO_RIGID_xxxx) **Note that a carriage return is not added**. See also DeformableToRigid.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for DeformableToRigid dtor:

```
var key = dtor.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the DeformableToRigid. **Note that a carriage return is not added**. See also DeformableToRigid.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for DeformableToRigid dtor:

```
var cards = dtor.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last deformable to rigid in the model.

Arguments

- **Model** (Model)

Model to get last deformable to rigid in

Returns

DeformableToRigid object (or null if there are no deformable to rigids in the model).

Return type

DeformableToRigid

Example

To get the last deformable to rigid in model m:

```
var dtor = DeformableToRigid.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free deformable to rigid label in the model. Also see DeformableToRigid.FirstFreeLabel(), DeformableToRigid.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free deformable to rigid label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

DeformableToRigid label.

Return type

Number

Example

To get the last free deformable to rigid label in model m:

```
var label = DeformableToRigid.LastFreeLabel(m);
```

Next()

Description

Returns the next deformable to rigid in the model.

Arguments

No arguments

Returns

DeformableToRigid object (or null if there are no more deformable to rigids in the model).

Return type

DeformableToRigid

Example

To get the deformable to rigid in model m after deformable to rigid dtor:

```
var dtor = dtor.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) deformable to rigid label in the model. Also see DeformableToRigid.FirstFreeLabel(), DeformableToRigid.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free deformable to rigid label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

DeformableToRigid label.

Return type

Number

Example

To get the next free deformable to rigid label in model m:

```
var label = DeformableToRigid.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a deformable to rigid.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only deformable to rigids from that model can be picked. If the argument is a Flag then only deformable to rigids that are flagged with *limit* can be selected. If omitted, or null, any deformable to rigids from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

DeformableToRigid object (or null if not picked)

Return type

DeformableToRigid

Example

To pick a deformable to rigid from model m giving the prompt 'Pick deformable to rigid from screen':

```
var dtor = DeformableToRigid.Pick('Pick deformable to rigid from screen', m);
```

Previous()

Description

Returns the previous deformable to rigid in the model.

Arguments

No arguments

Returns

DeformableToRigid object (or null if there are no more deformable to rigids in the model).

Return type

DeformableToRigid

Example

To get the deformable to rigid in model m before deformable to rigid dtor:

```
var dtor = dtor.Previous();
```

RemoveDefToRegAutoCard(ctype*[integer]*, index*[integer]*)

Description

Removes the D2R or R2D cards for *DEFORMABLE_TO_RIGID_AUTOMATC.

Arguments

- **ctype** (integer)

The card type you want removed. Can be D2R or R2D.

- **index** (integer)

The card index you want removed. **Note that card indices start at 0, not 1**.

Returns

No return value.

Example

To remove the D2R card data for the 3rd D2R card from Deformable to Rigid dtor:

```
if (dtor.d2r >= 3)
{
    var dtor_data = dtor.RemoveDefToRegAutoCard(DeformableToRigid.D2R, 2);
}
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the deformable to rigids in the model.

Arguments

- **Model** (Model)

Model that all deformable to rigids will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the deformable to rigids in model m, from 1000000:

```
DeformableToRigid.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged deformable to rigids in the model.

Arguments

- **Model** (Model)

Model that all the flagged deformable to rigids will be renumbered in

- **flag** (Flag)

Flag set on the deformable to rigids that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the deformable to rigids in model m flagged with f, from 1000000:

```
DeformableToRigid.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select deformable to rigids using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting deformable to rigids

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only deformable to rigids from that model can be selected. If the argument is a Flag then only deformable to rigids that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any deformable to rigids can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of deformable to rigids selected or null if menu cancelled

Return type

Number

Example

To select deformable to rigids from model m, flagging those selected with flag f, giving the prompt 'Select deformable to rigids':

```
DeformableToRigid.Select(f, 'Select deformable to rigids', m);
```

To select deformable to rigids, flagging those selected with flag f but limiting selection to deformable to rigids flagged with flag l, giving the prompt 'Select deformable to rigids':

```
DeformableToRigid.Select(f, 'Select deformable to rigids', l);
```

SetDefToRegAutoCard(ctype*[integer]*, index*[integer]*, ptype*[integer]*, pid*[integer]*, lrb (optional)*[integer]*)

Description

Sets the D2r or R2D card data f*DEFORMABLE_TO_RIGID_AUTOMATIC.

Arguments

- **ctype** (integer)

The card type you want to set. Can be D2R or R2D.

- **index** (integer)

The D2R or R2D card index you want to set. **Note that cards start at 0, not 1**.

- **ptype** (integer)

Part type (PTYPE). Can be DeformableToRigid.PART or DeformableToRigid.PSET.

- **pid** (integer)

Part or Part Set ID.

- **lrb (optional)** (integer)

LRB Part ID (only for card type D2R)

Returns

No return value.

Example

To set the 3rd D2R card to ptype DeformabletoRigid.PART, pid 100 and lrb 200, for DeformableToRigid dtor:

```
dtor.SetDefToRegAutoCard(DeformabletoRigid.D2R, 2, DeformabletoRigid.PART, 100,
200);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the deformable to rigid.

Arguments

- **flag** (Flag)

Flag to set on the deformable to rigid

Returns

No return value

Example

To set flag f for deformable to rigid dtor:

```
dtor.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the deformable to rigid. The deformable to rigid will be sketched until you either call DeformableToRigid.Unsketch(), DeformableToRigid.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the deformable to rigid is sketched. If omitted redraw is true. If you want to sketch several deformable to rigids and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch deformable to rigid dtor:

```
dtor.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged deformable to rigids in the model. The deformable to rigids will be sketched until you either call DeformableToRigid.Unsketch(), DeformableToRigid.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged deformable to rigids will be sketched in

- **flag** (Flag)

Flag set on the deformable to rigids that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the deformable to rigids are sketched. If omitted redraw is true. If you want to sketch flagged deformable to rigids several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all deformable to rigids flagged with flag in model m:

```
DeformableToRigid.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of deformable to rigids in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing deformable to rigids should be counted. If false or omitted referenced but undefined deformable to rigids will also be included in the total.

Returns

number of deformable to rigids

Return type

Number

Example

To get the total number of deformable to rigids in model m:

```
var total = DeformableToRigid.Total(m);
```

Unblank()

Description

Unblanks the deformable to rigid

Arguments

No arguments

Returns

No return value

Example

To unblank deformable to rigid dtor:

```
dtor.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the deformable to rigids in the model.

Arguments

- **Model** (Model)

Model that all deformable to rigids will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the deformable to rigids in model m:

```
DeformableToRigid.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged deformable to rigids in the model.

Arguments

- **Model** (Model)

Model that the flagged deformable to rigids will be unblanked in

- **flag** (Flag)

Flag set on the deformable to rigids that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the deformable to rigids in model m flagged with f:

```
DeformableToRigid.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the deformable to rigids in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all deformable to rigids will be unset in

- **flag** (Flag)

Flag to unset on the deformable to rigids

Returns

No return value

Example

To unset the flag f on all the deformable to rigids in model m:

```
DeformableToRigid.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the deformable to rigid.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the deformable to rigid is unsketched. If omitted redraw is true. If you want to unsketch several deformable to rigids and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch deformable to rigid dtor:

```
dtor.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all deformable to rigids.

Arguments

- **Model** (Model)

Model that all deformable to rigids will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the deformable to rigids are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all deformable to rigids in model m:

```
DeformableToRigid.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged deformable to rigids in the model.

Arguments

- **Model** (Model)

Model that all deformable to rigids will be unsketched in

- **flag** (Flag)

Flag set on the deformable to rigids that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the deformable to rigids are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all deformable to rigids flagged with flag in model m:

```
DeformableToRigid.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

DeformableToRigid object.

Return type

DeformableToRigid

Example

To check if DeformableToRigid property dtor.example is a parameter by using the DeformableToRigid.GetParameter() method:

```
if (dtor.ViewParameters().GetParameter(dtor.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for deformable to rigid with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The deformable to rigid object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for deformable to rigid dtor:

```
dtor.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for deformable to rigid dtor:

```
dtor.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for deformable to rigid. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for deformable to rigid dtor:

```
dtor.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this deformable to rigid.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for deformable to rigid dtor:

```
var xrefs = dtor.Xrefs();
```

---

toString()

Description

Creates a string containing the DeformableToRigid data in keyword format. Note that this contains the keyword header and the keyword cards. See also DeformableToRigid.Keyword() and DeformableToRigid.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for DeformableToRigid dtor in keyword format

```
var i_str = dtor.toString();
```

Accelerometer class

The Accelerometer class gives you access to seatbelt accelerometer cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- ExtractColour()
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)

- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

Accelerometer properties

Name	Type	Description
colour	Colour	The colour of the accelerometer
exists (read only)	logical	true if accelerometer exists, false if referred to but not defined.
igrav	integer	Gravitational acceleration due to body force loads is included in acceleration output if igrav is 0, removed if igrav is 1.
include	integer	The Include file number that the accelerometer is in.
intopt	integer	Integration option; velocities are integrated from global accelerations and transformed into local system if intopt is 0, they are integrated directly from local accelerations if intopt is 1.
label	integer	Accelerometer number. Also see the sbacid property which is an alternative name for this.
mass	real	Optional added mass for accelerometer
model (read only)	integer	The Model number that the accelerometer is in.
nid1	integer	Node number 1
nid2	integer	Node number 2
nid3	integer	Node number 3
sbacid	integer	Accelerometer number. Also see the label property which is an alternative name for this.
transparency	integer	The transparency of the accelerometer (0-100) 0% is opaque, 100% is transparent.

Detailed Description

The Accelerometer class allows you to create, modify, edit and manipulate seatbelt accelerometer cards. See the documentation below for more details.

Constructor

new Accelerometer(Model*[Model]*, sbacid*[integer]*, nid1*[integer]*, nid2*[integer]*, nid3*[integer]*, igrav (optional)*[integer]*, intopt (optional)*[integer]*, mass (optional)*[real]*)

Description

Create a new Seatbelt Accelerometer object.

Arguments

- **Model** (Model)

Model that accelerometer will be created in

- **sbacid** (integer)

Accelerometer number. Also see the label property which is an alternative name for this.

- **nid1** (integer)

Node number 1

- **nid2** (integer)

Node number 2

- **nid3** (integer)

Node number 3

- **igrav (optional)** (integer)

Gravitational acceleration due to body force loads is included in acceleration output if igrav is 0, removed if igrav is 1.

- **intopt (optional)** (integer)

Integration option; velocities are integrated from global accelerations and transformed into local system if intopt is 0, they are integrated directly from local accelerations if intopt is 1.

- **mass (optional)** (real)

Optional added mass for accelerometer

Returns

Accelerometer object

Return type

Accelerometer

Example

To create a new seatbelt accelerometer in model m with label 100, nodes 1, 2 and 3:

```
var a = new Accelerometer(m, 100, 1, 2, 3);
```

Details of functions

AssociateComment(Comment[*Comment*])

Description

Associates a comment with a accelerometer.

Arguments

- **Comment** (Comment)

Comment that will be attached to the accelerometer

Returns

No return value

Example

To associate comment c to the accelerometer a:

```
a.AssociateComment(c);
```

Blank()

Description

Blanks the accelerometer

Arguments

No arguments

Returns

No return value

Example

To blank accelerometer a:

```
a.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the accelerometers in the model.

Arguments

- **Model** (Model)

Model that all accelerometers will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the accelerometers in model m:

```
Accelerometer.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged accelerometers in the model.

Arguments

- **Model** (Model)

Model that all the flagged accelerometers will be blanked in

- **flag** (Flag)

Flag set on the accelerometers that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the accelerometers in model m flagged with f:

```
Accelerometer.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the accelerometer is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if accelerometer a is blanked:

```
if (a.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse accelerometer a:

```
a.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the accelerometer.

Arguments

- **flag** (Flag)

Flag to clear on the accelerometer

Returns

No return value

Example

To clear flag f for accelerometer a:

```
a.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the accelerometer. The target include of the copied accelerometer can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

Accelerometer object

Return type

Accelerometer

Example

To copy accelerometer a into accelerometer z:

```
var z = a.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a accelerometer

Arguments

- **Model** (Model)

Model that the accelerometer will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

Accelerometer object (or null if not made).

Return type

Accelerometer

Example

To start creating a accelerometer in model m:

```
var a = Accelerometer.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a accelerometer.

Arguments

- **Comment** (Comment)

Comment that will be detached from the accelerometer

Returns

No return value

Example

To detach comment c from the accelerometer a:

```
a.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit accelerometer a:

```
a.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for accelerometer with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description

autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The accelerometer object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for accelerometer a:

```
a.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for accelerometer a:

```
a.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for accelerometer. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for accelerometer a:

```
a.Error("My custom error");
```

ExtractColour()

Description

Extracts the **actual** colour used for accelerometer.
By default in PRIMER many entities such as elements get their colour automatically from the part that they are in. PRIMER cycles through 13 default colours based on the label of the entity. In this case the accelerometer colour property will return the value Colour.PART instead of the actual colour. This method will return the actual colour which is used for drawing the accelerometer.

Arguments

No arguments

Returns

colour value (integer)

Return type

Number

Example

To return the colour used for drawing accelerometer a:

```
var colour = a.ExtractColour();
```

First(Model*[Model]*) [static]

Description

Returns the first accelerometer in the model.

Arguments

- **Model** (Model)

Model to get first accelerometer in

Returns

Accelerometer object (or null if there are no accelerometers in the model).

Return type

Accelerometer

Example

To get the first accelerometer in model m:

```
var a = Accelerometer.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free accelerometer label in the model. Also see Accelerometer.LastFreeLabel(), Accelerometer.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free accelerometer label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

Accelerometer label.

Return type

Number

Example

To get the first free accelerometer label in model m:

```
var label = Accelerometer.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the accelerometers in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all accelerometers will be flagged in

- **flag** (Flag)

Flag to set on the accelerometers

Returns

No return value

Example

To flag all of the accelerometers with flag f in model m:

```
Accelerometer.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the accelerometer is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the accelerometer

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if accelerometer a has flag f set on it:

```
if (a.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each accelerometer in the model.
Note that ForEach has been designed to make looping over accelerometers as fast as possible and so has some limitations.
Firstly, a single temporary Accelerometer object is created and on each function call it is updated with the current accelerometer data. This means that you should not try to store the Accelerometer object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new accelerometers inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all accelerometers are in

- **func** (function)

Function to call for each accelerometer

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the accelerometers in model m:

```
Accelerometer.ForEach(m, test);
function test(a)
{
// a is Accelerometer object
}
```

To call function test for all of the accelerometers in model m with optional object:

```
var data = { x:0, y:0 };
Accelerometer.ForEach(m, test, data);
function test(a, extra)
{
// a is Accelerometer object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Accelerometer objects or properties for all of the accelerometers in a model in PRIMER. If the optional property argument is not given then an array of Accelerometer objects is returned. If the property argument is given, that property value for each accelerometer is returned in the array instead of a Accelerometer object

Arguments

- **Model** (Model)

Model to get accelerometers from

- **property (optional)** (string)

Name for property to get for all accelerometers in the model

Returns

Array of Accelerometer objects or properties

Return type

Array

Example

To make an array of Accelerometer objects for all of the accelerometers in model m:

```
var a = Accelerometer.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each accelerometer in model m:

```
var a = Accelerometer.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a accelerometer.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the accelerometer a:

```
var comm_array = a.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of Accelerometer objects for all of the flagged accelerometers in a model in PRIMER If the optional property argument is not given then an array of Accelerometer objects is returned. If the property argument is given, then that property value for each accelerometer is returned in the array instead of a Accelerometer object

Arguments

- **Model** (Model)

Model to get accelerometers from

- **flag** (Flag)

Flag set on the accelerometers that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged accelerometers in the model

Returns

Array of Accelerometer objects or properties

Return type

Array

Example

To make an array of Accelerometer objects for all of the accelerometers in model m flagged with f:

```
var a = Accelerometer.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the accelerometers in model m flagged with f:

```
var a = Accelerometer.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the Accelerometer object for a accelerometer ID.

Arguments

- **Model** (Model)

Model to find the accelerometer in

- **number** (integer)

number of the accelerometer you want the Accelerometer object for

Returns

Accelerometer object (or null if accelerometer does not exist).

Return type

Accelerometer

Example

To get the Accelerometer object for accelerometer 100 in model m

```
var a = Accelerometer.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a Accelerometer property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the Accelerometer.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

accelerometer property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if Accelerometer property a.example is a parameter:

```
Options.property_parameter_names = true;
if (a.GetParameter(a.example) ) do_something...
Options.property_parameter_names = false;
```

To check if Accelerometer property a.example is a parameter by using the GetParameter method:

```
if (a.ViewParameters().GetParameter(a.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this accelerometer (*ELEMENT_SEATBELT_ACCELEROMETER) **Note that a carriage return is not added**. See also Accelerometer.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for accelerometer a:

```
var key = a.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the accelerometer. **Note that a carriage return is not added**. See also Accelerometer.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for accelerometer a:

```
var cards = a.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last accelerometer in the model.

Arguments

- **Model** (Model)

Model to get last accelerometer in

Returns

Accelerometer object (or null if there are no accelerometers in the model).

Return type

Accelerometer

Example

To get the last accelerometer in model m:

```
var a = Accelerometer.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free accelerometer label in the model. Also see Accelerometer.FirstFreeLabel(), Accelerometer.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free accelerometer label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

Accelerometer label.

Return type

Number

Example

To get the last free accelerometer label in model m:

```
var label = Accelerometer.LastFreeLabel(m);
```

Next()

Description

Returns the next accelerometer in the model.

Arguments

No arguments

Returns

Accelerometer object (or null if there are no more accelerometers in the model).

Return type

Accelerometer

Example

To get the accelerometer in model m after accelerometer a:

```
var a = a.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) accelerometer label in the model. Also see Accelerometer.FirstFreeLabel(), Accelerometer.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free accelerometer label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

Accelerometer label.

Return type

Number

Example

To get the next free accelerometer label in model m:

```
var label = Accelerometer.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a accelerometer.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only accelerometers from that model can be picked. If the argument is a Flag then only accelerometers that are flagged with *limit* can be selected. If omitted, or null, any accelerometers from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

Accelerometer object (or null if not picked)

Return type

Accelerometer

Example

To pick a accelerometer from model m giving the prompt 'Pick accelerometer from screen':

```
var a = Accelerometer.Pick('Pick accelerometer from screen', m);
```

Previous()

Description

Returns the previous accelerometer in the model.

Arguments

No arguments

Returns

Accelerometer object (or null if there are no more accelerometers in the model).

Return type

Accelerometer

Example

To get the accelerometer in model m before accelerometer a:

```
var a = a.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the accelerometers in the model.

Arguments

- **Model** (Model)

Model that all accelerometers will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the accelerometers in model m, from 1000000:

```
Accelerometer.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged accelerometers in the model.

Arguments

- **Model** (Model)

Model that all the flagged accelerometers will be renumbered in

- **flag** (Flag)

Flag set on the accelerometers that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the accelerometers in model m flagged with f, from 1000000:

```
Accelerometer.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select accelerometers using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting accelerometers

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only accelerometers from that model can be selected. If the argument is a Flag then only accelerometers that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any accelerometers can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of accelerometers selected or null if menu cancelled

Return type

Number

Example

To select accelerometers from model m, flagging those selected with flag f, giving the prompt 'Select accelerometers':

```
Accelerometer.Select(f, 'Select accelerometers', m);
```

To select accelerometers, flagging those selected with flag f but limiting selection to accelerometers flagged with flag l, giving the prompt 'Select accelerometers':

```
Accelerometer.Select(f, 'Select accelerometers', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the accelerometer.

Arguments

- **flag** (Flag)

Flag to set on the accelerometer

Returns

No return value

Example

To set flag f for accelerometer a:

```
a.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the accelerometer. The accelerometer will be sketched until you either call Accelerometer.Unsketch(), Accelerometer.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the accelerometer is sketched. If omitted redraw is true. If you want to sketch several accelerometers and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch accelerometer a:

```
a.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged accelerometers in the model. The accelerometers will be sketched until you either call Accelerometer.Unsketch(), Accelerometer.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged accelerometers will be sketched in

- **flag** (Flag)

Flag set on the accelerometers that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the accelerometers are sketched. If omitted redraw is true. If you want to sketch flagged accelerometers several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all accelerometers flagged with flag in model m:

```
Accelerometer.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of accelerometers in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing accelerometers should be counted. If false or omitted referenced but undefined accelerometers will also be included in the total.

Returns

number of accelerometers

Return type

Number

Example

To get the total number of accelerometers in model m:

```
var total = Accelerometer.Total(m);
```

Unblank()

Description

Unblanks the accelerometer

Arguments

No arguments

Returns

No return value

Example

To unblank accelerometer a:

```
a.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the accelerometers in the model.

Arguments

- **Model** (Model)

Model that all accelerometers will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the accelerometers in model m:

```
Accelerometer.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged accelerometers in the model.

Arguments

- **Model** (Model)

Model that the flagged accelerometers will be unblanked in

- **flag** (Flag)

Flag set on the accelerometers that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the accelerometers in model m flagged with f:

```
Accelerometer.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the accelerometers in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all accelerometers will be unset in

- **flag** (Flag)

Flag to unset on the accelerometers

Returns

No return value

Example

To unset the flag f on all the accelerometers in model m:

```
Accelerometer.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the accelerometer.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the accelerometer is unsketched. If omitted redraw is true. If you want to unsketch several accelerometers and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch accelerometer a:

```
a.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all accelerometers.

Arguments

- **Model** (Model)

Model that all accelerometers will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the accelerometers are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all accelerometers in model m:

```
Accelerometer.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged accelerometers in the model.

Arguments

- **Model** (Model)

Model that all accelerometers will be unsketched in

- **flag** (Flag)

Flag set on the accelerometers that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the accelerometers are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all accelerometers flagged with flag in model m:

```
Accelerometer.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

Accelerometer object.

Return type

Accelerometer

Example

To check if Accelerometer property a.example is a parameter by using the Accelerometer.GetParameter() method:

```
if (a.ViewParameters().GetParameter(a.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for accelerometer with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The accelerometer object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for accelerometer a:

```
a.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for accelerometer a:

```
a.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for accelerometer. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for accelerometer a:

```
a.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this accelerometer.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for accelerometer a:

```
var xrefs = a.Xrefs();
```

---

toString()

Description

Creates a string containing the accelerometer data in keyword format. Note that this contains the keyword header and the keyword cards. See also Accelerometer.Keyword() and Accelerometer.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for accelerometer a in keyword format

```
var str = a.toString();
```

Beam class

The Beam class gives you access to beam cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- FindBeamInBox(Model*[Model]*, xmin*[real]*, xmax*[real]*, ymin*[real]*, ymax*[real]*, zmin*[real]*, zmax*[real]*, flag (optional)*[integer]*, excl (optional)*[integer]*, vis_only (optional)*[integer]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- ElemCut(Database cross section label*[integer]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- ExtractColour()
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SectionFacePoints(face*[integer]*)
- SectionFaces()
- SectionPoints()

- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- TiedNodeCheck(Contact label*[integer]*, Flag*[Flag]*, Option1*[integer]*, Option2*[integer]*)
- Timestep()
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

Beam properties

Name	Type	Description
cid	integer	Coordinate system ID (_SCALAR)
cid	integer	Coordinate system ID at node 1 (_SCALR)
colour	Colour	The colour of the beam
d1	real	Section parameter 1
d2	real	Section parameter 2
d3	real	Section parameter 3
d4	real	Section parameter 4
d5	real	Section parameter 5
d6	real	Section parameter 6
dofn1	integer	Active degree of freedom at node 1 (_SCALAR)
dofn2	integer	Active degree of freedom at node 2 (_SCALAR)
dofns	integer	Active degrees of freedom at nodes 1 and 2 (_SCALR)
eid	integer	Beam number. Also see the label property which is an alternative name for this.
elbow	logical	If ELBOW option is set. Can be true or false
exists (read only)	logical	true if beam exists, false if referred to but not defined.
include	integer	The Include file number that the beam is in.
iner	real	Mass moment of inertia for beam
label	integer	Beam number. Also see the eid property which is an alternative name for this.
local	integer	Coordinate system option
mn	integer	Middle Node for elbow beam
model (read only)	integer	The Model number that the beam is in.
n1	integer	Node number 1
n2	integer	Node number 2
n3	integer	Node number 3
nodes (read only)	integer	Number of nodes beam has
offset	real	If _OFFSET option is set. Can be true or false
orientation	real	If _ORIENTATION option is set. Can be true or false
parm1	real	Thickness parameter 1
parm2	real	Thickness parameter 2

parm3	real	Thickness parameter 3
parm4	real	Thickness parameter 4
parm5	real	Thickness parameter 5
pid	integer	Part number
pid1	integer	Part number 1 for spotweld beam
pid2	integer	Part number 2 for spotweld beam
pid_opt	logical	If _PID option is set. Can be true or false
rr1	integer	Rotational release code at node 1
rr2	integer	Rotational release code at node 2
rt1	integer	Translational release code at node 1
rt2	integer	Translational release code at node 2
scalar	logical	If _SCALAR option is set. Can be true or false
scalr	logical	If _SCALR option is set. Can be true or false
section	logical	If _SECTION option is set. Can be true or false
sn1	integer	Scalar Node number 1
sn2	integer	Scalar Node number 2
stype	string	Section type
thickness	logical	If _THICKNESS option is set. Can be true or false
transparency	integer	The transparency of the beam (0-100) 0% is opaque, 100% is transparent.
vol	real	Volume of beam
vx	real	Orientation vector X at node 1
vy	real	Orientation vector Y at node 1
vz	real	Orientation vector Z at node 1
warpage	logical	If WARPAGE option is set. Can be true or false
wx1	real	Offset vector X at node 1
wx2	real	Offset vector X at node 2
wy1	real	Offset vector Y at node 1
wy2	real	Offset vector Y at node 2
wz1	real	Offset vector Z at node 1
wz2	real	Offset vector Z at node 2

Detailed Description

The Beam class allows you to create, modify, edit and manipulate beam cards. See the documentation below for more details.

Constructor

new Beam(Model*[Model]*, eid*[integer]*, pid*[integer]*, n1*[integer]*, n2 (optional)*[integer]*, n3 (optional)*[integer]*)

Description

Create a new Beam object. Use either 1, 2 or 3 nodes when creating a new beam.

Arguments

- **Model** (Model)

Model that beam will be created in

- **eid** (integer)

Beam number

- **pid** (integer)

Part number

- **n1** (integer)

Node number 1

- **n2 (optional)** (integer)

Node number 2

- **n3 (optional)** (integer)

Node number 3

Returns

Beam object

Return type

Beam

Example

To create a new beam in model m with label 100, part 10 and nodes 1, 2, 3:

```
var b = new Beam(m, 100, 10, 1, 2, 3);
```

Details of functions

AssociateComment(Comment[*Comment*])

Description

Associates a comment with a beam.

Arguments

- **Comment** (Comment)

Comment that will be attached to the beam

Returns

No return value

Example

To associate comment c to the beam b:

```
b.AssociateComment(c);
```

Blank()

Description

Blanks the beam

Arguments

No arguments

Returns

No return value

Example

To blank beam b:

```
b.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the beams in the model.

Arguments

- **Model** (Model)

Model that all beams will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the beams in model m:

```
Beam.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged beams in the model.

Arguments

- **Model** (Model)

Model that all the flagged beams will be blanked in

- **flag** (Flag)

Flag set on the beams that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the beams in model m flagged with f:

```
Beam.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the beam is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if beam b is blanked:

```
if (b.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse beam b:

```
b.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the beam.

Arguments

- **flag** (Flag)

Flag to clear on the beam

Returns

No return value

Example

To clear flag f for beam b:

```
b.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the beam. The target include of the copied beam can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

Beam object

Return type

Beam

Example

To copy beam b into beam z:

```
var z = b.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a beam

Arguments

- **Model** (Model)

Model that the beam will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

Beam object (or null if not made).

Return type

Beam

Example

To start creating a beam in model m:

```
var b = Beam.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a beam.

Arguments

- **Comment** (Comment)

Comment that will be detached from the beam

Returns

No return value

Example

To detach comment c from the beam b:

```
b.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit beam b:

```
b.Edit();
```

ElemCut(Database cross section label*[integer]*)

Description

Returns coordinates of the intersections between a beam and a database cross section.
Note this function does not check that the beam is in the cross section definition (part set)

Arguments

- **Database cross section label** (integer)

The label of the database cross section.

Returns

An array containing the x,y,z coordinates of the cut point, or NULL if it does not cut

Return type

Array

Example

To get the cut line coordinates between database cross section 200 and beam b:

```
var data = b.ElemCut(200)
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for beam with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The beam object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for beam b:

```
b.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for beam b:

```
b.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for beam. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for beam b:

```
b.Error("My custom error");
```

ExtractColour()

Description

Extracts the **actual** colour used for beam.
By default in PRIMER many entities such as elements get their colour automatically from the part that they are in. PRIMER cycles through 13 default colours based on the label of the entity. In this case the beam colour property will return the value Colour.PART instead of the actual colour. This method will return the actual colour which is used for drawing the beam.

Arguments

No arguments

Returns

colour value (integer)

Return type

Number

Example

To return the colour used for drawing beam b:

```
var colour = b.ExtractColour();
```

FindBeamInBox(Model*[Model]*, xmin*[real]*, xmax*[real]*, ymin*[real]*, ymax*[real]*, zmin*[real]*, zmax*[real]*, flag (optional)*[integer]*, excl (optional)*[integer]*, vis_only (optional)*[integer]*) [static]

Description

Returns an array of Beam objects for the beams within a box. Please note this function provides a list of all beams that could potentially be in the box (using computationally cheap bounding box comparison) it is not a rigorous test of whether the beam is actually in the box. Note an extension of "spot_thickness" is applied to each beam. This may include beams that are ostensibly outside box. The user should apply their own test. (this function is intended to provide an upper bound of elems to test) Setting the "excl" flag will require that the beam is fully contained, but this may not capture all the beams you want to process.

Arguments

- **Model** (Model)

Model designated model

- **xmin** (real)

Minimum bound in global x

- **xmax** (real)

Maximum bound in global x

- **ymin** (real)

Minimum bound in global y

- **ymax** (real)

Maximum bound in global y

- **zmin** (real)

Minimum bound in global z

- **zmax** (real)

Maximum bound in global z

- **flag (optional)** (integer)

Optional flag to restrict beams considered, if 0 all beams considered

- **excl (optional)** (integer)

Optional flag (0) Apply inclusive selection (1) Apply exclusive selection inclusive selection means elements intersect box exclusive selection means elements contained in box

- **vis_only (optional)** (integer)

Optional flag to consider visible elements only (1), if (0) all elements considered

Returns

Array of Beam objects

Return type

Array

Example

To get an array of Beam objects for flagged beams within defined box (inclusive selection)

```
      var s = Beam.FindBeamInBox(m, xmin, xmax, ymin, ymax, zmin, zmax, flag, 0,
0);
      if(s.length) ...
```

First(Model[*Model*]) [static]

Description

Returns the first beam in the model.

Arguments

- **Model** (Model)

Model to get first beam in

Returns

Beam object (or null if there are no beams in the model).

Return type

Beam

Example

To get the first beam in model m:

```
var b = Beam.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free beam label in the model. Also see Beam.LastFreeLabel(), Beam.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free beam label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

Beam label.

Return type

Number

Example

To get the first free beam label in model m:

```
var label = Beam.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the beams in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all beams will be flagged in

- **flag** (Flag)

Flag to set on the beams

Returns

No return value

Example

To flag all of the beams with flag f in model m:

```
Beam.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the beam is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the beam

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if beam b has flag f set on it:

```
if (b.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each beam in the model.
Note that ForEach has been designed to make looping over beams as fast as possible and so has some limitations. Firstly, a single temporary Beam object is created and on each function call it is updated with the current beam data. This means that you should not try to store the Beam object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new beams inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all beams are in

- **func** (function)

Function to call for each beam

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the beams in model m:

```
Beam.ForEach(m, test);
function test(b)
{
// b is Beam object
}
```

To call function test for all of the beams in model m with optional object:

```
var data = { x:0, y:0 };
Beam.ForEach(m, test, data);
function test(b, extra)
{
// b is Beam object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Beam objects or properties for all of the beams in a model in PRIMER. If the optional property argument is not given then an array of Beam objects is returned. If the property argument is given, that property value for each beam is returned in the array instead of a Beam object

Arguments

- **Model** (Model)

Model to get beams from

- **property (optional)** (string)

Name for property to get for all beams in the model

Returns

Array of Beam objects or properties

Return type

Array

Example

To make an array of Beam objects for all of the beams in model m:

```
var a = Beam.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each beam in model m:

```
var a = Beam.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a beam.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the beam b:

```
var comm_array = b.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of Beam objects for all of the flagged beams in a model in PRIMER If the optional property argument is not given then an array of Beam objects is returned. If the property argument is given, then that property value for each beam is returned in the array instead of a Beam object

Arguments

- **Model** (Model)

Model to get beams from

- **flag** (Flag)

Flag set on the beams that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged beams in the model

Returns

Array of Beam objects or properties

Return type

Array

Example

To make an array of Beam objects for all of the beams in model m flagged with f:

```
var b = Beam.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the beams in model m flagged with f:

```
var a = Beam.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the Beam object for a beam ID.

Arguments

- **Model** (Model)

Model to find the beam in

- **number** (integer)

number of the beam you want the Beam object for

Returns

Beam object (or null if beam does not exist).

Return type

Beam

Example

To get the Beam object for beam 100 in model m

```
var b = Beam.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a Beam property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the Beam.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

beam property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if Beam property b.example is a parameter:

```
Options.property_parameter_names = true;
if (b.GetParameter(b.example) ) do_something...
Options.property_parameter_names = false;
```

To check if Beam property b.example is a parameter by using the GetParameter method:

```
if (b.ViewParameters().GetParameter(b.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this beam (*BEAM, *BEAM_SCALAR or *BEAM_SCALAR_VALUE). **Note that a carriage return is not added**. See also Beam.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for beam s:

```
var key = s.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the beam. **Note that a carriage return is not added**. See also Beam.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for beam b:

```
var cards = b.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last beam in the model.

Arguments

- **Model** (Model)

Model to get last beam in

Returns

Beam object (or null if there are no beams in the model).

Return type

Beam

Example

To get the last beam in model m:

```
var b = Beam.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free beam label in the model. Also see Beam.FirstFreeLabel(), Beam.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free beam label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

Beam label.

Return type

Number

Example

To get the last free beam label in model m:

```
var label = Beam.LastFreeLabel(m);
```

Next()

Description

Returns the next beam in the model.

Arguments

No arguments

Returns

Beam object (or null if there are no more beams in the model).

Return type

Beam

Example

To get the beam in model m after beam b:

```
var b = b.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) beam label in the model. Also see Beam.FirstFreeLabel(), Beam.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free beam label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

Beam label.

Return type

Number

Example

To get the next free beam label in model m:

```
var label = Beam.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a beam.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only beams from that model can be picked. If the argument is a Flag then only beams that are flagged with *limit* can be selected. If omitted, or null, any beams from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

Beam object (or null if not picked)

Return type

Beam

Example

To pick a beam from model m giving the prompt 'Pick beam from screen':

```
var b = Beam.Pick('Pick beam from screen', m);
```

Previous()

Description

Returns the previous beam in the model.

Arguments

No arguments

Returns

Beam object (or null if there are no more beams in the model).

Return type

Beam

Example

To get the beam in model m before beam b:

```
var b = b.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the beams in the model.

Arguments

- **Model** (Model)

Model that all beams will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the beams in model m, from 1000000:

```
Beam.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged beams in the model.

Arguments

- **Model** (Model)

Model that all the flagged beams will be renumbered in

- **flag** (Flag)

Flag set on the beams that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the beams in model m flagged with f, from 1000000:

```
Beam.RenumberFlagged(m, f, 1000000);
```

SectionFacePoints(face*[integer]*)

Description

Returns the indices of the points for a faces to plot the true section of the beam. Note face numbers start at 0. Beam.SectionPoints must be called before this method.

Arguments

- **face** (integer)

Face to get indices for

Returns

Array of integers

Return type

Number

Example

To get the indices of the points for the second face on beam b:

```
var indices = b.SectionFacePoints(1);
```

SectionFaces()

Description

Returns the number of faces to plot the true section of the beam. Beam.SectionPoints must be called before this method.

Arguments

No arguments

Returns

integer

Return type

Number

Example

To get the number of faces for beam b:

```
var faces = b.SectionFaces();
```

SectionPoints()

Description

Returns the point coordinates to plot the true section of the beam. They are returned in a single array of numbers.

Arguments

No arguments

Returns

Array of reals

Return type

Number

Example

To get the point coordinates for beam b:

```
var points = b.SectionPoints();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select beams using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting beams

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only beams from that model can be selected. If the argument is a Flag then only beams that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any beams can be

selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of beams selected or null if menu cancelled

Return type

Number

Example

To select beams from model m, flagging those selected with flag f, giving the prompt 'Select beams':

```
Beam.Select(f, 'Select beams', m);
```

To select beams, flagging those selected with flag f but limiting selection to beams flagged with flag l, giving the prompt 'Select beams':

```
Beam.Select(f, 'Select beams', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the beam.

Arguments

- **flag** (Flag)

Flag to set on the beam

Returns

No return value

Example

To set flag f for beam b:

```
b.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the beam. The beam will be sketched until you either call Beam.Unsketch(), Beam.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the beam is sketched. If omitted redraw is true. If you want to sketch several beams and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch beam b:

```
b.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged beams in the model. The beams will be sketched until you either call Beam.Unsketch(), Beam.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged beams will be sketched in

- **flag** (Flag)

Flag set on the beams that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the beams are sketched. If omitted redraw is true. If you want to sketch flagged beams several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all beams flagged with flag in model m:

```
Beam.SketchFlagged(m, flag);
```

TiedNodeCheck(Contact label*[integer]*, Flag*[Flag]*, Option1*[integer]*, Option2*[integer]*)

Description

Checks if nodes of beam are tied by contact or directly attached (non-zero option1)

Arguments

- **Contact label** (integer)

The label of the tied contact. If zero the tied contact is found for the beam by reverse lookup.

- **Flag** (Flag)

flag bit

- **Option1** (integer)

Directly tied node (logical OR) 0:NONE 1:NRB/C_EXNO 2:BEAM 4:SHELL 8:SOLID 16:TSHELL

- **Option2** (integer)

0:No action 1:report error if directly attached node (acc. option1) also captured by contact

Returns

string

Return type

String

Example

To check if both nodes of beam b are tied by contact 200 or attach directly to constraint, beam or shell:

```
var message = b.TiedNodeCheck(200, flag, 1|2|4, 1)
```

Timestep()

Description

Calculates the timestep for the beam

Arguments

No arguments

Returns

real

Return type

Number

Example

To calculate the timestep for beam b:

```
var timestep = b.Timestep();
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of beams in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing beams should be counted. If false or omitted referenced but undefined beams will also be included in the total.

Returns

number of beams

Return type

Number

Example

To get the total number of beams in model m:

```
var total = Beam.Total(m);
```

Unblank()

Description

Unblanks the beam

Arguments

No arguments

Returns

No return value

Example

To unblank beam b:

```
b.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the beams in the model.

Arguments

- **Model** (Model)

Model that all beams will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the beams in model m:

```
Beam.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged beams in the model.

Arguments

- **Model** (Model)

Model that the flagged beams will be unblanked in

- **flag** (Flag)

Flag set on the beams that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the beams in model m flagged with f:

```
Beam.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the beams in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all beams will be unset in

- **flag** (Flag)

Flag to unset on the beams

Returns

No return value

Example

To unset the flag f on all the beams in model m:

```
Beam.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the beam.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the beam is unsketched. If omitted redraw is true. If you want to unsketch several beams and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch beam b:

```
b.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all beams.

Arguments

- **Model** (Model)

Model that all beams will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the beams are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all beams in model m:

```
Beam.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged beams in the model.

Arguments

- **Model** (Model)

Model that all beams will be unsketched in

- **flag** (Flag)

Flag set on the beams that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the beams are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all beams flagged with flag in model m:

```
Beam.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with ’method chaining’ (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

Beam object.

Return type

Beam

Example

To check if Beam property b.example is a parameter by using the Beam.GetParameter() method:

```
if (b.ViewParameters().GetParameter(b.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for beam with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The beam object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for beam b:

```
b.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for beam b:

```
b.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for beam. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for beam b:

```
b.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this beam.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for beam b:

```
var xrefs = b.Xrefs();
```

---

toString()

Description

Creates a string containing the beam data in keyword format. Note that this contains the keyword header and the keyword cards. See also Beam.Keyword() and Beam.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for beam b in keyword format

```
var str = b.toString();
```

Discrete class

The Discrete class gives you access to element discrete cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- ExtractColour()
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Timestep()
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()

- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

Discrete properties

Name	Type	Description
colour	Colour	The colour of the discrete
eid	integer	Discrete number. Also see the label property which is an alternative name for this.
exists (read only)	logical	true if discrete exists, false if referred to but not defined.
include	integer	The Include file number that the discrete is in.
label	integer	Discrete number. Also see the eid property which is an alternative name for this.
lcid	integer	Loadcurve for offset vs time
lciddr	integer	Loadcurve for offset vs time during dynamic relaxation
lco	boolean	If LCO option is set. Can be true or false
model (read only)	integer	The Model number that the discrete is in.
n1	integer	Node number 1
n2	integer	Node number 2
offset	real	Initial offset
pf	integer	Print flag. Set to write forces to the DEFORC file
pid	integer	Part number
s	real	Scale factor on forces
transparency	integer	The transparency of the discrete (0-100) 0% is opaque, 100% is transparent.
vid	integer	Orientation vector

Detailed Description

The Discrete class allows you to create, modify, edit and manipulate discrete cards. See the documentation below for more details.

Constructor

new Discrete(Model*[Model]*, eid*[integer]*, pid*[integer]*, n1*[integer]*, n2*[integer]*, vid (optional)*[integer]*, s (optional)*[real]*, pf (optional)*[integer]*, offset (optional)*[real]*)

Description

Create a new Discrete object.

Arguments

- **Model** (Model)

Model that discrete will be created in

- **eid** (integer)

Discrete number

- **pid** (integer)

Part number

- **n1** (integer)

Node number 1

- **n2** (integer)

Node number 2

- **vid (optional)** (integer)

Orientation vector

- **s (optional)** (real)

Scale factor on forces

- **pf (optional)** (integer)

Print flag. Set to write forces to the DEFORC file

- **offset (optional)** (real)

Initial offset

Returns

Discrete object

Return type

Discrete

Example

To create a new discrete in model m with label 200, in part 10, on nodes 1 and 2

```
var m = new Discrete(m, 200, 10, 1, 2);
```

Details of functions

AssociateComment(Comment[*Comment*])

Description

Associates a comment with a discrete.

Arguments

- **Comment** (Comment)

Comment that will be attached to the discrete

Returns

No return value

Example

To associate comment c to the discrete d:

```
d.AssociateComment(c);
```

Blank()

Description

Blanks the discrete

Arguments

No arguments

Returns

No return value

Example

To blank discrete d:

```
d.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the discretes in the model.

Arguments

- **Model** (Model)

Model that all discretes will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the discretes in model m:

```
Discrete.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged discretes in the model.

Arguments

- **Model** (Model)

Model that all the flagged discretes will be blanked in

- **flag** (Flag)

Flag set on the discretes that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the discretes in model m flagged with f:

```
Discrete.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the discrete is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if discrete d is blanked:

```
if (d.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse discrete d:

```
d.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the discrete.

Arguments

- **flag** (Flag)

Flag to clear on the discrete

Returns

No return value

Example

To clear flag f for discrete d:

```
d.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the discrete. The target include of the copied discrete can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

Discrete object

Return type

Discrete

Example

To copy discrete d into discrete z:

```
var z = d.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a discrete

Arguments

- **Model** (Model)

Model that the discrete will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

Discrete object (or null if not made).

Return type

Discrete

Example

To start creating a discrete in model m:

```
var d = Discrete.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a discrete.

Arguments

- **Comment** (Comment)

Comment that will be detached from the discrete

Returns

No return value

Example

To detach comment c from the discrete d:

```
d.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit discrete d:

```
d.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for discrete with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description

autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The discrete object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for discrete d:

```
d.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for discrete d:

```
d.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for discrete. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for discrete d:

```
d.Error("My custom error");
```

ExtractColour()

Description

Extracts the **actual** colour used for discrete.
By default in PRIMER many entities such as elements get their colour automatically from the part that they are in. PRIMER cycles through 13 default colours based on the label of the entity. In this case the discrete colour property will return the value Colour.PART instead of the actual colour. This method will return the actual colour which is used for drawing the discrete.

Arguments

No arguments

Returns

colour value (integer)

Return type

Number

Example

To return the colour used for drawing discrete d:

```
var colour = d.ExtractColour();
```

First(Model*[Model]*) [static]

Description

Returns the first discrete in the model.

Arguments

- **Model** (Model)

Model to get first discrete in

Returns

Discrete object (or null if there are no discretes in the model).

Return type

Discrete

Example

To get the first discrete in model m:

```
var d = Discrete.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free discrete label in the model. Also see Discrete.LastFreeLabel(), Discrete.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free discrete label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

Discrete label.

Return type

Number

Example

To get the first free discrete label in model m:

```
var label = Discrete.FirstFreeLabel(m);
```

FlagAll(Model[*Model*], flag[*Flag*]) [static]

Description

Flags all of the discretes in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all discretes will be flagged in

- **flag** (Flag)

Flag to set on the discretes

Returns

No return value

Example

To flag all of the discretes with flag f in model m:

```
Discrete.FlagAll(m, f);
```

Flagged(flag[*Flag*])

Description

Checks if the discrete is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the discrete

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if discrete d has flag f set on it:

```
if (d.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each discrete in the model.
Note that ForEach has been designed to make looping over discretes as fast as possible and so has some limitations.
Firstly, a single temporary Discrete object is created and on each function call it is updated with the current discrete data. This means that you should not try to store the Discrete object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new discretes inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all discretes are in

- **func** (function)

Function to call for each discrete

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the discretes in model m:

```
Discrete.ForEach(m, test);
function test(d)
{
// d is Discrete object
}
```

To call function test for all of the discretes in model m with optional object:

```
var data = { x:0, y:0 };
Discrete.ForEach(m, test, data);
function test(d, extra)
{
// d is Discrete object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Discrete objects or properties for all of the discretes in a model in PRIMER. If the optional property argument is not given then an array of Discrete objects is returned. If the property argument is given, that property value for each discrete is returned in the array instead of a Discrete object

Arguments

- **Model** (Model)

Model to get discretes from

- **property (optional)** (string)

Name for property to get for all discretes in the model

Returns

Array of Discrete objects or properties

Return type

Array

Example

To make an array of Discrete objects for all of the discretes in model m:

```
var a = Discrete.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each discrete in model m:

```
var a = Discrete.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a discrete.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the discrete d:

```
var comm_array = d.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of Discrete objects for all of the flagged discretes in a model in PRIMER If the optional property argument is not given then an array of Discrete objects is returned. If the property argument is given, then that property value for each discrete is returned in the array instead of a Discrete object

Arguments

- **Model** (Model)

Model to get discretes from

- **flag** (Flag)

Flag set on the discretes that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged discretes in the model

Returns

Array of Discrete objects or properties

Return type

Array

Example

To make an array of Discrete objects for all of the discretes in model m flagged with f:

```
var d = Discrete.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the discretes in model m flagged with f:

```
var a = Discrete.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the Discrete object for a discrete ID.

Arguments

- **Model** (Model)

Model to find the discrete in

- **number** (integer)

number of the discrete you want the Discrete object for

Returns

Discrete object (or null if discrete does not exist).

Return type

Discrete

Example

To get the Discrete object for discrete 100 in model m

```
var d = Discrete.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a Discrete property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the Discrete.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

discrete property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if Discrete property d.example is a parameter:

```
Options.property_parameter_names = true;
if (d.GetParameter(d.example) ) do_something...
Options.property_parameter_names = false;
```

To check if Discrete property d.example is a parameter by using the GetParameter method:

```
if (d.ViewParameters().GetParameter(d.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this discrete (*ELEMENT_DISCRETE). **Note that a carriage return is not added**. See also Discrete.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for discrete m:

```
var key = m.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the discrete. **Note that a carriage return is not added**. See also Discrete.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for discrete d:

```
var cards = d.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last discrete in the model.

Arguments

- **Model** (Model)

Model to get last discrete in

Returns

Discrete object (or null if there are no discretes in the model).

Return type

Discrete

Example

To get the last discrete in model m:

```
var d = Discrete.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free discrete label in the model. Also see Discrete.FirstFreeLabel(), Discrete.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free discrete label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

Discrete label.

Return type

Number

Example

To get the last free discrete label in model m:

```
var label = Discrete.LastFreeLabel(m);
```

Next()

Description

Returns the next discrete in the model.

Arguments

No arguments

Returns

Discrete object (or null if there are no more discretes in the model).

Return type

Discrete

Example

To get the discrete in model m after discrete d:

```
var d = d.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) discrete label in the model. Also see Discrete.FirstFreeLabel(), Discrete.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free discrete label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

Discrete label.

Return type

Number

Example

To get the next free discrete label in model m:

```
var label = Discrete.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a discrete.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only discretes from that model can be picked. If the argument is a Flag then only discretes that are flagged with *limit* can be selected. If omitted, or null, any discretes from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

Discrete object (or null if not picked)

Return type

Discrete

Example

To pick a discrete from model m giving the prompt 'Pick discrete from screen':

```
var d = Discrete.Pick('Pick discrete from screen', m);
```

Previous()

Description

Returns the previous discrete in the model.

Arguments

No arguments

Returns

Discrete object (or null if there are no more discretes in the model).

Return type

Discrete

Example

To get the discrete in model m before discrete d:

```
var d = d.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the discretes in the model.

Arguments

- **Model** (Model)

Model that all discretes will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the discretes in model m, from 1000000:

```
Discrete.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged discretes in the model.

Arguments

- **Model** (Model)

Model that all the flagged discretes will be renumbered in

- **flag** (Flag)

Flag set on the discretes that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the discretes in model m flagged with f, from 1000000:

```
Discrete.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select discretes using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting discretes

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only discretes from that model can be selected. If the argument is a Flag then only discretes that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any discretes can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of discretes selected or null if menu cancelled

Return type

Number

Example

To select discretes from model m, flagging those selected with flag f, giving the prompt 'Select discretes':

```
Discrete.Select(f, 'Select discretes', m);
```

To select discretes, flagging those selected with flag f but limiting selection to discretes flagged with flag l, giving the prompt 'Select discretes':

```
Discrete.Select(f, 'Select discretes', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the discrete.

Arguments

- **flag** (Flag)

Flag to set on the discrete

Returns

No return value

Example

To set flag f for discrete d:

```
d.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the discrete. The discrete will be sketched until you either call Discrete.Unsketch(), Discrete.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the discrete is sketched. If omitted redraw is true. If you want to sketch several discretes and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch discrete d:

```
d.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged discretes in the model. The discretes will be sketched until you either call Discrete.Unsketch(), Discrete.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged discretes will be sketched in

- **flag** (Flag)

Flag set on the discretes that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the discretes are sketched. If omitted redraw is true. If you want to sketch flagged discretes several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all discretes flagged with flag in model m:

```
Discrete.SketchFlagged(m, flag);
```

Timestep()

Description

Calculates the timestep for the discrete

Arguments

No arguments

Returns

real

Return type

Number

Example

To calculate the timestep for discrete d:

```
var timestep = d.Timestep();
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of discretes in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing discretes should be counted. If false or omitted referenced but undefined discretes will also be included in the total.

Returns

number of discretes

Return type

Number

Example

To get the total number of discretes in model m:

```
var total = Discrete.Total(m);
```

Unblank()

Description

Unblanks the discrete

Arguments

No arguments

Returns

No return value

Example

To unblank discrete d:

```
d.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the discretes in the model.

Arguments

- **Model** (Model)

Model that all discretes will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the discretes in model m:

```
Discrete.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged discretes in the model.

Arguments

- **Model** (Model)

Model that the flagged discretes will be unblanked in

- **flag** (Flag)

Flag set on the discretes that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the discretes in model m flagged with f:

```
Discrete.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the discretes in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all discretes will be unset in

- **flag** (Flag)

Flag to unset on the discretes

Returns

No return value

Example

To unset the flag f on all the discretes in model m:

```
Discrete.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the discrete.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the discrete is unsketched. If omitted redraw is true. If you want to unsketch several discretes and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch discrete d:

```
d.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all discretes.

Arguments

- **Model** (Model)

Model that all discretes will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the discretes are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all discretes in model m:

```
Discrete.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged discretes in the model.

Arguments

- **Model** (Model)

Model that all discretes will be unsketched in

- **flag** (Flag)

Flag set on the discretes that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the discretes are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all discretes flagged with flag in model m:

```
Discrete.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

Discrete object.

Return type

Discrete

Example

To check if Discrete property d.example is a parameter by using the Discrete.GetParameter() method:

```
if (d.ViewParameters().GetParameter(d.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for discrete with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The discrete object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for discrete d:

```
d.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for discrete d:

```
d.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for discrete. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for discrete d:

```
d.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this discrete.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for discrete d:

```
var xrefs = d.Xrefs();
```

---

toString()

Description

Creates a string containing the discrete data in keyword format. Note that this contains the keyword header and the keyword cards. See also Discrete.Keyword() and Discrete.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for discrete d in keyword format

```
var s = d.toString();
```

DiscreteSphere class

The DiscreteSphere class gives you access to element discrete sphere cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- ExtractColour()
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()

- toString()

DiscreteSphere properties

Name	Type	Description
colour	Colour	The colour of the discrete sphere
exists (read only)	logical	true if discrete sphere exists, false if referred to but not defined.
idist	integer	Distribution of DES properties.
include	integer	The Include file number that the discrete sphere is in.
inertia	real	Mass moment of inertia.
isd_mi	real	Standard deviation for mass moment of inertia or inertia per unit density(depending on whether the _VOLUME option is set), if idist is -1 or 1. Shape parameter for mass moment of inertia or inertia per unit density(depending on whether the _VOLUME option is set), if idist is -2 or 2.
mass	real	Mass or volume value (depending on whether the _VOLUME option is set).
model (read only)	integer	The Model number that the discrete sphere is in.
msd_mm	real	Standard deviation for mass or volume(depending on whether the _VOLUME option is set), if idist is -1 or 1. Shape parameter for mass or volume(depending on whether the _VOLUME option is set), if idist is -2 or 2.
nid	integer	Node ID.
pid	integer	Part ID to which this element belongs.
radius	real	Particle radius.
rsd_mr	real	Standard deviation for radius, if idist is -1 or 1. Shape parameter for radius, if idist is -2 or 2.
transparency	integer	The transparency of the discrete sphere (0-100) 0% is opaque, 100% is transparent.
volume	logical	Turns _VOLUME on or OFF. Note that this does NOT refer to the data field VOLUME. For the latter see the mass property.

Detailed Description

The DiscreteSphere class allows you to create, modify, edit and manipulate discrete sphere cards. See the documentation below for more details.

Constructor

new DiscreteSphere(Model*[Model]*, nid*[integer]*, pid*[integer]*, mass*[real]*, inertia*[real]*, radius*[real]*)

Description

Create a new DiscreteSphere object.

Arguments

- **Model** (Model)

Model that discrete sphere will be created in

- **nid** (integer)

Node ID and Element ID are the same for discrete spheres.

- **pid** (integer)

Part ID to which this element belongs.

- **mass** (real)

Mass or volume value.

- **inertia** (real)

Mass moment of inertia.

- **radius** (real)

Particle radius.

Returns

DiscreteSphere object

Return type

DiscreteSphere

Example

To create a new discrete sphere in model m with nid = 100, pid = 400, mass = 0.9, inertia = 2.5, radius = 2.0:

```
var dsph = new DiscreteSphere(m, 100, 400, 0.9, 2.5, 2.0);
```

Details of functions

AssociateComment(Comment[*Comment*])

Description

Associates a comment with a discrete sphere.

Arguments

- **Comment** (Comment)

Comment that will be attached to the discrete sphere

Returns

No return value

Example

To associate comment c to the discrete sphere dsph:

```
dsph.AssociateComment(c);
```

Blank()

Description

Blanks the discrete sphere

Arguments

No arguments

Returns

No return value

Example

To blank discrete sphere dsph:

```
dsph.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the discrete spheres in the model.

Arguments

- **Model** (Model)

Model that all discrete spheres will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the discrete spheres in model m:

```
DiscreteSphere.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged discrete spheres in the model.

Arguments

- **Model** (Model)

Model that all the flagged discrete spheres will be blanked in

- **flag** (Flag)

Flag set on the discrete spheres that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the discrete spheres in model m flagged with f:

```
DiscreteSphere.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the discrete sphere is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if discrete sphere dsph is blanked:

```
if (dsph.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse discrete sphere dsph:

```
dsph.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the discrete sphere.

Arguments

- **flag** (Flag)

Flag to clear on the discrete sphere

Returns

No return value

Example

To clear flag f for discrete sphere dsph:

```
dsph.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the discrete sphere. The target include of the copied discrete sphere can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

DiscreteSphere object

Return type

DiscreteSphere

Example

To copy discrete sphere dsph into discrete sphere z:

```
var z = dsph.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a discrete sphere

Arguments

- **Model** (Model)

Model that the discrete sphere will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

DiscreteSphere object (or null if not made).

Return type

DiscreteSphere

Example

To start creating a discrete sphere in model m:

```
var dsph = DiscreteSphere.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a discrete sphere.

Arguments

- **Comment** (Comment)

Comment that will be detached from the discrete sphere

Returns

No return value

Example

To detach comment c from the discrete sphere dsph:

```
dsph.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit discrete sphere dsph:

```
dsph.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for discrete sphere with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description

autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The discrete sphere object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for discrete sphere dsph:

```
dsph.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for discrete sphere dsph:

```
dsph.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for discrete sphere. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for discrete sphere dsph:

```
dsph.Error("My custom error");
```

ExtractColour()

Description

Extracts the **actual** colour used for discrete sphere.
By default in PRIMER many entities such as elements get their colour automatically from the part that they are in. PRIMER cycles through 13 default colours based on the label of the entity. In this case the discrete sphere colour property will return the value Colour.PART instead of the actual colour. This method will return the actual colour which is used for drawing the discrete sphere.

Arguments

No arguments

Returns

colour value (integer)

Return type

Number

Example

To return the colour used for drawing discrete sphere dsph:

```
var colour = dsph.ExtractColour();
```

First(Model*[Model]*) [static]

Description

Returns the first discrete sphere in the model.

Arguments

- **Model** (Model)

Model to get first discrete sphere in

Returns

DiscreteSphere object (or null if there are no discrete spheres in the model).

Return type

DiscreteSphere

Example

To get the first discrete sphere in model m:

```
var dsph = DiscreteSphere.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free discrete sphere label in the model. Also see DiscreteSphere.LastFreeLabel(), DiscreteSphere.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free discrete sphere label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

DiscreteSphere label.

Return type

Number

Example

To get the first free discrete sphere label in model m:

```
var label = DiscreteSphere.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the discrete spheres in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all discrete spheres will be flagged in

- **flag** (Flag)

Flag to set on the discrete spheres

Returns

No return value

Example

To flag all of the discrete spheres with flag f in model m:

```
DiscreteSphere.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the discrete sphere is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the discrete sphere

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if discrete sphere dsph has flag f set on it:

```
if (dsph.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each discrete sphere in the model.
Note that ForEach has been designed to make looping over discrete spheres as fast as possible and so has some limitations.
Firstly, a single temporary DiscreteSphere object is created and on each function call it is updated with the current discrete sphere data. This means that you should not try to store the DiscreteSphere object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new discrete spheres inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all discrete spheres are in

- **func** (function)

Function to call for each discrete sphere

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the discrete spheres in model m:

```
DiscreteSphere.ForEach(m, test);
function test(dsph)
{
// dsph is DiscreteSphere object
}
```

To call function test for all of the discrete spheres in model m with optional object:

```
var data = { x:0, y:0 };
DiscreteSphere.ForEach(m, test, data);
function test(dsph, extra)
{
// dsph is DiscreteSphere object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of DiscreteSphere objects or properties for all of the discrete spheres in a model in PRIMER. If the optional property argument is not given then an array of DiscreteSphere objects is returned. If the property argument is given, that property value for each discrete sphere is returned in the array instead of a DiscreteSphere object

Arguments

- **Model** (Model)

Model to get discrete spheres from

- **property (optional)** (string)

Name for property to get for all discrete spheres in the model

Returns

Array of DiscreteSphere objects or properties

Return type

Array

Example

To make an array of DiscreteSphere objects for all of the discrete spheres in model m:

```
var a = DiscreteSphere.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each discrete sphere in model m:

```
var a = DiscreteSphere.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a discrete sphere.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the discrete sphere dsph:

```
var comm_array = dsph.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of DiscreteSphere objects for all of the flagged discrete spheres in a model in PRIMER If the optional property argument is not given then an array of DiscreteSphere objects is returned. If the property argument is given, then that property value for each discrete sphere is returned in the array instead of a DiscreteSphere object

Arguments

- **Model** (Model)

Model to get discrete spheres from

- **flag** (Flag)

Flag set on the discrete spheres that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged discrete spheres in the model

Returns

Array of DiscreteSphere objects or properties

Return type

Array

Example

To make an array of DiscreteSphere objects for all of the discrete spheres in model m flagged with f:

```
var dsph = DiscreteSphere.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the discrete spheres in model m flagged with f:

```
var a = DiscreteSphere.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the DiscreteSphere object for a discrete sphere ID.

Arguments

- **Model** (Model)

Model to find the discrete sphere in

- **number** (integer)

number of the discrete sphere you want the DiscreteSphere object for

Returns

DiscreteSphere object (or null if discrete sphere does not exist).

Return type

DiscreteSphere

Example

To get the DiscreteSphere object for discrete sphere 100 in model m

```
var dsph = DiscreteSphere.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a DiscreteSphere property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the DiscreteSphere.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

discrete sphere property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if DiscreteSphere property dsph.example is a parameter:

```
Options.property_parameter_names = true;
if (dsph.GetParameter(dsph.example) ) do_something...
Options.property_parameter_names = false;
```

To check if DiscreteSphere property dsph.example is a parameter by using the GetParameter method:

```
if (dsph.ViewParameters().GetParameter(dsph.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this discrete sphere (*ELEMENT_DISCRETE_SPHERE or *ELEMENT_DISCRETE_SPHERE_VOLUME). **Note that a carriage return is not added**. See also DiscreteSphere.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for discrete sphere dsph:

```
var key = dsph.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the discrete sphere. **Note that a carriage return is not added**. See also DiscreteSphere.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for discrete sphere dsph:

```
var cards = dsph.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last discrete sphere in the model.

Arguments

- **Model** (Model)

Model to get last discrete sphere in

Returns

DiscreteSphere object (or null if there are no discrete spheres in the model).

Return type

DiscreteSphere

Example

To get the last discrete sphere in model m:

```
var dsph = DiscreteSphere.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free discrete sphere label in the model. Also see DiscreteSphere.FirstFreeLabel(), DiscreteSphere.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free discrete sphere label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

DiscreteSphere label.

Return type

Number

Example

To get the last free discrete sphere label in model m:

```
var label = DiscreteSphere.LastFreeLabel(m);
```

Next()

Description

Returns the next discrete sphere in the model.

Arguments

No arguments

Returns

DiscreteSphere object (or null if there are no more discrete spheres in the model).

Return type

DiscreteSphere

Example

To get the discrete sphere in model m after discrete sphere dsph:

```
var dsph = dsph.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) discrete sphere label in the model. Also see DiscreteSphere.FirstFreeLabel(), DiscreteSphere.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free discrete sphere label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

DiscreteSphere label.

Return type

Number

Example

To get the next free discrete sphere label in model m:

```
var label = DiscreteSphere.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a discrete sphere.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only discrete spheres from that model can be picked. If the argument is a Flag then only discrete spheres that are flagged with *limit* can be selected. If omitted, or null, any discrete spheres from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

DiscreteSphere object (or null if not picked)

Return type

DiscreteSphere

Example

To pick a discrete sphere from model m giving the prompt 'Pick discrete sphere from screen':

```
var dsph = DiscreteSphere.Pick('Pick discrete sphere from screen', m);
```

Previous()

Description

Returns the previous discrete sphere in the model.

Arguments

No arguments

Returns

DiscreteSphere object (or null if there are no more discrete spheres in the model).

Return type

DiscreteSphere

Example

To get the discrete sphere in model m before discrete sphere dsph:

```
var dsph = dsph.Previous();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select discrete spheres using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting discrete spheres

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only discrete spheres from that model can be selected. If the argument is a Flag then only discrete spheres that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any discrete spheres can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of discrete spheres selected or null if menu cancelled

Return type

Number

Example

To select discrete spheres from model m, flagging those selected with flag f, giving the prompt 'Select discrete spheres':

```
DiscreteSphere.Select(f, 'Select discrete spheres', m);
```

To select discrete spheres, flagging those selected with flag f but limiting selection to discrete spheres flagged with flag l, giving the prompt 'Select discrete spheres':

```
DiscreteSphere.Select(f, 'Select discrete spheres', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the discrete sphere.

Arguments

- **flag** (Flag)

Flag to set on the discrete sphere

Returns

No return value

Example

To set flag f for discrete sphere dsph:

```
dsph.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the discrete sphere. The discrete sphere will be sketched until you either call DiscreteSphere.Unsketch(), DiscreteSphere.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the discrete sphere is sketched. If omitted redraw is true. If you want to sketch several discrete spheres and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch discrete sphere dsph:

```
dsph.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged discrete spheres in the model. The discrete spheres will be sketched until you either call DiscreteSphere.Unsketch(), DiscreteSphere.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged discrete spheres will be sketched in

- **flag** (Flag)

Flag set on the discrete spheres that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the discrete spheres are sketched. If omitted redraw is true. If you want to sketch flagged discrete spheres several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all discrete spheres flagged with flag in model m:

```
DiscreteSphere.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of discrete spheres in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing discrete spheres should be counted. If false or omitted referenced but undefined discrete spheres will also be included in the total.

Returns

number of discrete spheres

Return type

Number

Example

To get the total number of discrete spheres in model m:

```
var total = DiscreteSphere.Total(m);
```

Unblank()

Description

Unblanks the discrete sphere

Arguments

No arguments

Returns

No return value

Example

To unblank discrete sphere dsph:

```
dsph.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the discrete spheres in the model.

Arguments

- **Model** (Model)

Model that all discrete spheres will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the discrete spheres in model m:

```
DiscreteSphere.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged discrete spheres in the model.

Arguments

- **Model** (Model)

Model that the flagged discrete spheres will be unblanked in

- **flag** (Flag)

Flag set on the discrete spheres that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the discrete spheres in model m flagged with f:

```
DiscreteSphere.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the discrete spheres in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all discrete spheres will be unset in

- **flag** (Flag)

Flag to unset on the discrete spheres

Returns

No return value

Example

To unset the flag f on all the discrete spheres in model m:

```
DiscreteSphere.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the discrete sphere.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the discrete sphere is unsketched. If omitted redraw is true. If you want to unsketch several discrete spheres and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch discrete sphere dsph:

```
dsph.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all discrete spheres.

Arguments

- **Model** (Model)

Model that all discrete spheres will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the discrete spheres are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all discrete spheres in model m:

```
DiscreteSphere.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged discrete spheres in the model.

Arguments

- **Model** (Model)

Model that all discrete spheres will be unsketched in

- **flag** (Flag)

Flag set on the discrete spheres that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the discrete spheres are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all discrete spheres flagged with flag in model m:

```
DiscreteSphere.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

DiscreteSphere object.

Return type

DiscreteSphere

Example

To check if DiscreteSphere property dsph.example is a parameter by using the DiscreteSphere.GetParameter() method:

```
if (dsph.ViewParameters().GetParameter(dsph.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for discrete sphere with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The discrete sphere object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for discrete sphere dsph:

```
dsph.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for discrete sphere dsph:

```
dsph.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for discrete sphere. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for discrete sphere dsph:

```
dsph.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this discrete sphere.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for discrete sphere dsph:

```
var xrefs = dsph.Xrefs();
```

---

toString()

Description

Creates a string containing the discrete sphere data in keyword format. Note that this contains the keyword header and the keyword cards. See also DiscreteSphere.Keyword() and DiscreteSphere.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for discrete sphere dsph in keyword format

```
var s = dsph.toString();
```

Mass class

The Mass class gives you access to element mass cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- ExtractColour()
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)

- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

Mass constants

Name	Description
Mass.NODE_SET	Mass is *MASS_NODE_SET.

Mass properties

Name	Type	Description
colour	Colour	The colour of the mass
eid	integer	Mass number. Also see the label property which is an alternative name for this.
exists (read only)	logical	true if mass exists, false if referred to but not defined.
id	integer	Node id or node set id
include	integer	The Include file number that the mass is in.
label	integer	Mass number. Also see the eid property which is an alternative name for this.
mass	real	Mass value
model (read only)	integer	The Model number that the mass is in.
node_set	integer	The type of the mass. Can be false (*MASS) or Mass.NODE_SET (*MASS_NODE_SET)
pid	integer	Part ID
transparency	integer	The transparency of the mass (0-100) 0% is opaque, 100% is transparent.

Detailed Description

The Mass class allows you to create, modify, edit and manipulate mass cards. See the documentation below for more details.

Constructor

new Mass(Model*[Model]*, eid*[integer]*, id*[integer]*, mass*[real]*, node set (optional)*[integer]*)

Description

Create a new Mass object.

Arguments

- **Model** (Model)

Model that mass will be created in

- **eid** (integer)

Mass number

- **id** (integer)

Node id or node set id

- **mass** (real)

Mass value

- **node set (optional)** (integer)

Only used if a node set is used

Returns

Mass object

Return type

Mass

Example

To create a new mass in model m with label 200, on node 500, or node set 500, with a mass of 3.5, use one of the following:

```
var m = new Mass(m, 200, 500, 3.5);

var m = new Mass(m, 200, 500, 3.5, Mass.NODE_SET);
```

Details of functions

AssociateComment(Comment[*Comment*])

Description

Associates a comment with a mass.

Arguments

- **Comment** (Comment)

Comment that will be attached to the mass

Returns

No return value

Example

To associate comment c to the mass m:

```
m.AssociateComment(c);
```

Blank()

Description

Blanks the mass

Arguments

No arguments

Returns

No return value

Example

To blank mass m:

```
m.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the masss in the model.

Arguments

- **Model** (Model)

Model that all masss will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the masss in model m:

```
Mass.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged masss in the model.

Arguments

- **Model** (Model)

Model that all the flagged masss will be blanked in

- **flag** (Flag)

Flag set on the masss that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the masss in model m flagged with f:

```
Mass.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the mass is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if mass m is blanked:

```
if (m.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse mass m:

```
m.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the mass.

Arguments

- **flag** (Flag)

Flag to clear on the mass

Returns

No return value

Example

To clear flag f for mass m:

```
m.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the mass. The target include of the copied mass can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

Mass object

Return type

Mass

Example

To copy mass m into mass z:

```
var z = m.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a mass

Arguments

- **Model** (Model)

Model that the mass will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

Mass object (or null if not made).

Return type

Mass

Example

To start creating a mass in model m:

```
var m = Mass.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a mass.

Arguments

- **Comment** (Comment)

Comment that will be detached from the mass

Returns

No return value

Example

To detach comment c from the mass m:

```
m.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit mass m:

```
m.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for mass with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The mass object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for mass m:

```
m.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for mass m:

```
m.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for mass. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for mass m:

```
m.Error("My custom error");
```

ExtractColour()

Description

Extracts the **actual** colour used for mass.
By default in PRIMER many entities such as elements get their colour automatically from the part that they are in. PRIMER cycles through 13 default colours based on the label of the entity. In this case the mass colour property will return the value Colour.PART instead of the actual colour. This method will return the actual colour which is used for drawing the mass.

Arguments

No arguments

Returns

colour value (integer)

Return type

Number

Example

To return the colour used for drawing mass m:

```
var colour = m.ExtractColour();
```

First(Model*[Model]*) [static]

Description

Returns the first mass in the model.

Arguments

- **Model** (Model)

Model to get first mass in

Returns

Mass object (or null if there are no masss in the model).

Return type

Mass

Example

To get the first mass in model m:

```
var m = Mass.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free mass label in the model. Also see Mass.LastFreeLabel(), Mass.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free mass label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

Mass label.

Return type

Number

Example

To get the first free mass label in model m:

```
var label = Mass.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the masss in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all masss will be flagged in

- **flag** (Flag)

Flag to set on the masss

Returns

No return value

Example

To flag all of the masss with flag f in model m:

```
Mass.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the mass is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the mass

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if mass m has flag f set on it:

```
if (m.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each mass in the model.
Note that ForEach has been designed to make looping over masss as fast as possible and so has some limitations. Firstly, a single temporary Mass object is created and on each function call it is updated with the current mass data. This means that you should not try to store the Mass object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new masss inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all masss are in

- **func** (function)

Function to call for each mass

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the masss in model m:

```
Mass.ForEach(m, test);
function test(m)
{
// m is Mass object
}
```

To call function test for all of the masss in model m with optional object:

```
var data = { x:0, y:0 };
Mass.ForEach(m, test, data);
function test(m, extra)
{
// m is Mass object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Mass objects or properties for all of the masss in a model in PRIMER. If the optional property argument is not given then an array of Mass objects is returned. If the property argument is given, that property value for each mass is returned in the array instead of a Mass object

Arguments

- **Model** (Model)

Model to get masss from

- **property (optional)** (string)

Name for property to get for all masss in the model

Returns

Array of Mass objects or properties

Return type

Array

Example

To make an array of Mass objects for all of the masss in model m:

```
var a = Mass.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each mass in model m:

```
var a = Mass.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a mass.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the mass m:

```
var comm_array = m.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of Mass objects for all of the flagged masss in a model in PRIMER If the optional property argument is not given then an array of Mass objects is returned. If the property argument is given, then that property value for each mass is returned in the array instead of a Mass object

Arguments

- **Model** (Model)

Model to get masss from

- **flag** (Flag)

Flag set on the masss that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged masss in the model

Returns

Array of Mass objects or properties

Return type

Array

Example

To make an array of Mass objects for all of the masss in model m flagged with f:

```
var m = Mass.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the masss in model m flagged with f:

```
var a = Mass.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the Mass object for a mass ID.

Arguments

- **Model** (Model)

Model to find the mass in

- **number** (integer)

number of the mass you want the Mass object for

Returns

Mass object (or null if mass does not exist).

Return type

Mass

Example

To get the Mass object for mass 100 in model m

```
var m = Mass.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a Mass property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the Mass.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

mass property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if Mass property m.example is a parameter:

```
Options.property_parameter_names = true;
if (m.GetParameter(m.example) ) do_something...
Options.property_parameter_names = false;
```

To check if Mass property m.example is a parameter by using the GetParameter method:

```
if (m.ViewParameters().GetParameter(m.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this mass (*ELEMENT_MASS or *ELEMENT_MASS_NODE_SET). **Note that a carriage return is not added**. See also Mass.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for mass m:

```
var key = m.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the mass. **Note that a carriage return is not added**. See also Mass.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for mass m:

```
var cards = m.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last mass in the model.

Arguments

- **Model** (Model)

Model to get last mass in

Returns

Mass object (or null if there are no masss in the model).

Return type

Mass

Example

To get the last mass in model m:

```
var m = Mass.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free mass label in the model. Also see Mass.FirstFreeLabel(), Mass.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free mass label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

Mass label.

Return type

Number

Example

To get the last free mass label in model m:

```
var label = Mass.LastFreeLabel(m);
```

Next()

Description

Returns the next mass in the model.

Arguments

No arguments

Returns

Mass object (or null if there are no more masss in the model).

Return type

Mass

Example

To get the mass in model m after mass m:

```
var m = m.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) mass label in the model. Also see Mass.FirstFreeLabel(), Mass.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free mass label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

Mass label.

Return type

Number

Example

To get the next free mass label in model m:

```
var label = Mass.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a mass.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only masss from that model can be picked. If the argument is a Flag then only masss that are flagged with *limit* can be selected. If omitted, or null, any masss from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

Mass object (or null if not picked)

Return type

Mass

Example

To pick a mass from model m giving the prompt 'Pick mass from screen':

```
var m = Mass.Pick('Pick mass from screen', m);
```

Previous()

Description

Returns the previous mass in the model.

Arguments

No arguments

Returns

Mass object (or null if there are no more masss in the model).

Return type

Mass

Example

To get the mass in model m before mass m:

```
var m = m.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the masss in the model.

Arguments

- **Model** (Model)

Model that all masss will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the masss in model m, from 1000000:

```
Mass.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged masss in the model.

Arguments

- **Model** (Model)

Model that all the flagged masss will be renumbered in

- **flag** (Flag)

Flag set on the masss that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the masss in model m flagged with f, from 1000000:

```
Mass.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select masss using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting masss

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only masss from that model can be selected. If the argument is a Flag then only masss that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any masss can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of masss selected or null if menu cancelled

Return type

Number

Example

To select masss from model m, flagging those selected with flag f, giving the prompt 'Select masss':

```
Mass.Select(f, 'Select masss', m);
```

To select masss, flagging those selected with flag f but limiting selection to masss flagged with flag l, giving the prompt 'Select masss':

```
Mass.Select(f, 'Select masss', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the mass.

Arguments

- **flag** (Flag)

Flag to set on the mass

Returns

No return value

Example

To set flag f for mass m:

```
m.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the mass. The mass will be sketched until you either call Mass.Unsketch(), Mass.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the mass is sketched. If omitted redraw is true. If you want to sketch several masss and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch mass m:

```
m.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged masss in the model. The masss will be sketched until you either call Mass.Unsketch(), Mass.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged masss will be sketched in

- **flag** (Flag)

Flag set on the masss that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the masss are sketched. If omitted redraw is true. If you want to sketch flagged masss several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all masss flagged with flag in model m:

```
Mass.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of masss in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing masss should be counted. If false or omitted referenced but undefined masss will also be included in the total.

Returns

number of masss

Return type

Number

Example

To get the total number of masss in model m:

```
var total = Mass.Total(m);
```

Unblank()

Description

Unblanks the mass

Arguments

No arguments

Returns

No return value

Example

To unblank mass m:

```
m.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the masss in the model.

Arguments

- **Model** (Model)

Model that all masss will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the masss in model m:

```
Mass.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged masss in the model.

Arguments

- **Model** (Model)

Model that the flagged masss will be unblanked in

- **flag** (Flag)

Flag set on the masss that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the masss in model m flagged with f:

```
Mass.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the masss in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all masss will be unset in

- **flag** (Flag)

Flag to unset on the masss

Returns

No return value

Example

To unset the flag f on all the masss in model m:

```
Mass.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the mass.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the mass is unsketched. If omitted redraw is true. If you want to unsketch several masss and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch mass m:

```
m.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all masss.

Arguments

- **Model** (Model)

Model that all masss will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the masss are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all masss in model m:

```
Mass.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged masss in the model.

Arguments

- **Model** (Model)

Model that all masss will be unsketched in

- **flag** (Flag)

Flag set on the masss that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the masss are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all masss flagged with flag in model m:

```
Mass.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

Mass object.

Return type

Mass

Example

To check if Mass property m.example is a parameter by using the Mass.GetParameter() method:

```
if (m.ViewParameters().GetParameter(m.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for mass with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The mass object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for mass m:

```
m.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for mass m:

```
m.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for mass. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for mass m:

```
m.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this mass.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for mass m:

```
var xrefs = m.Xrefs();
```

---

toString()

Description

Creates a string containing the mass data in keyword format. Note that this contains the keyword header and the keyword cards. See also Mass.Keyword() and Mass.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for mass m in keyword format

```
var s = m.toString();
```

MassPart class

The MassPart class gives you access to element mass part cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- ExtractColour()
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

MassPart constants

Name	Description
MassPart.PART	Element is *ELEMENT_MASS_PART.
MassPart.SET	Element is *ELEMENT_MASS_PART_SET.

MassPart properties

Name	Type	Description
addmass	real	Added translational mass to be distributed to the nodes of the part or part set ID.
colour	Colour	The colour of the mass part
exists (read only)	logical	true if element mass part exists, false if referred to but not defined.
finmass	real	Final translational mass of the part or part set ID.
id	integer	Part or part set ID if the SET option is active.
include	integer	The Include file number that the element mass part is in.
lcid	integer	Optional load curve ID to scale the added mass at time = 0.
model (read only)	integer	The Model number that the element mass part is in.
mwd	integer	Optional flag for mass-weighted distribution.
option	constant	The Element Mass Part option. Can be MassPart.PART or MassPart.SET
transparency	integer	The transparency of the mass part (0-100) 0% is opaque, 100% is transparent.

Detailed Description

The MassPart class allows you to create, modify, edit and manipulate element mass part cards. See the documentation below for more details.

Constructor

new MassPart(Model*[Model]*, option*[constant]*, id*[integer]*, addmass (optional)*[real]*, finmass (optional)*[real]*)

Description

Create a new MassPart object.

Arguments

- **Model** (Model)

Model that mass part will be created in

- **option** (constant)

Suffix for element mass part. Can be MassPart.PART or MassPart.SET.

- **id** (integer)

Part or part set ID.

- **addmass (optional)** (real)

Added translational mass.

- **finmass (optional)** (real)

Final translational mass.

Returns

MassPart object

Return type

MassPart

Example

To create a new element mass part in model m with option _<BLANK> and part ID 10:

```
var mp = new MassPart(m, MassPart.PART, 10);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a element mass part.

Arguments

- **Comment** (Comment)

Comment that will be attached to the element mass part

Returns

No return value

Example

To associate comment c to the element mass part mp:

```
mp.AssociateComment(c);
```

Blank()

Description

Blanks the element mass part

Arguments

No arguments

Returns

No return value

Example

To blank element mass part mp:

```
mp.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the element mass parts in the model.

Arguments

- **Model** (Model)

Model that all element mass parts will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the element mass parts in model m:

```
MassPart.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged element mass parts in the model.

Arguments

- **Model** (Model)

Model that all the flagged element mass parts will be blanked in

- **flag** (Flag)

Flag set on the element mass parts that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the element mass parts in model m flagged with f:

```
MassPart.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the element mass part is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if element mass part mp is blanked:

```
if (mp.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse element mass part mp:

```
mp.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the element mass part.

Arguments

- **flag** (Flag)

Flag to clear on the element mass part

Returns

No return value

Example

To clear flag f for element mass part mp:

```
mp.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the element mass part. The target include of the copied element mass part can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

MassPart object

Return type

MassPart

Example

To copy element mass part mp into element mass part z:

```
var z = mp.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a element mass part

Arguments

- **Model** (Model)

Model that the element mass part will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

MassPart object (or null if not made).

Return type

MassPart

Example

To start creating a element mass part in model m:

```
var mp = MassPart.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a element mass part.

Arguments

- **Comment** (Comment)

Comment that will be detached from the element mass part

Returns

No return value

Example

To detach comment c from the element mass part mp:

```
mp.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit element mass part mp:

```
mp.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for element mass part with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The element mass part object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for element mass part mp:

```
mp.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for element mass part mp:

```
mp.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for element mass part. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for element mass part mp:

```
mp.Error("My custom error");
```

ExtractColour()

Description

Extracts the **actual** colour used for element mass part.
By default in PRIMER many entities such as elements get their colour automatically from the part that they are in. PRIMER cycles through 13 default colours based on the label of the entity. In this case the element mass part colour property will return the value Colour.PART instead of the actual colour. This method will return the actual colour which is used for drawing the element mass part.

Arguments

No arguments

Returns

colour value (integer)

Return type

Number

Example

To return the colour used for drawing element mass part mp:

```
var colour = mp.ExtractColour();
```

First(Model*[Model]*) [static]

Description

Returns the first element mass part in the model.

Arguments

- **Model** (Model)

Model to get first element mass part in

Returns

MassPart object (or null if there are no element mass parts in the model).

Return type

MassPart

Example

To get the first element mass part in model m:

```
var mp = MassPart.First(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the element mass parts in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all element mass parts will be flagged in

- **flag** (Flag)

Flag to set on the element mass parts

Returns

No return value

Example

To flag all of the element mass parts with flag f in model m:

```
MassPart.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the element mass part is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the element mass part

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if element mass part mp has flag f set on it:

```
if (mp.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each element mass part in the model.
Note that ForEach has been designed to make looping over element mass parts as fast as possible and so has some limitations.
Firstly, a single temporary MassPart object is created and on each function call it is updated with the current element mass part data. This means that you should not try to store the MassPart object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new element mass parts inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all element mass parts are in

- **func** (function)

Function to call for each element mass part

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the element mass parts in model m:

```
MassPart.ForEach(m, test);
function test(mp)
{
// mp is MassPart object
}
```

To call function test for all of the element mass parts in model m with optional object:

```
var data = { x:0, y:0 };
MassPart.ForEach(m, test, data);
function test(mp, extra)
{
// mp is MassPart object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of MassPart objects or properties for all of the element mass parts in a model in PRIMER. If the optional property argument is not given then an array of MassPart objects is returned. If the property argument is given, that property value for each element mass part is returned in the array instead of a MassPart object

Arguments

- **Model** (Model)

Model to get element mass parts from

- **property (optional)** (string)

Name for property to get for all element mass parts in the model

Returns

Array of MassPart objects or properties

Return type

Array

Example

To make an array of MassPart objects for all of the element mass parts in model m:

```
var a = MassPart.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each element mass part in model m:

```
var a = MassPart.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a element mass part.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the element mass part mp:

```
var comm_array = mp.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of MassPart objects for all of the flagged element mass parts in a model in PRIMER If the optional property argument is not given then an array of MassPart objects is returned. If the property argument is given, then that property value for each element mass part is returned in the array instead of a MassPart object

Arguments

- **Model** (Model)

Model to get element mass parts from

- **flag** (Flag)

Flag set on the element mass parts that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged element mass parts in the model

Returns

Array of MassPart objects or properties

Return type

Array

Example

To make an array of MassPart objects for all of the element mass parts in model m flagged with f:

```
var mp = MassPart.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the element mass parts in model m flagged with f:

```
var a = MassPart.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the MassPart object for a element mass part ID.

Arguments

- **Model** (Model)

Model to find the element mass part in

- **number** (integer)

number of the element mass part you want the MassPart object for

Returns

MassPart object (or null if element mass part does not exist).

Return type

MassPart

Example

To get the MassPart object for element mass part 100 in model m

```
var mp = MassPart.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a MassPart property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the MassPart.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

element mass part property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if MassPart property mp.example is a parameter:

```
Options.property_parameter_names = true;
if (mp.GetParameter(mp.example) ) do_something...
Options.property_parameter_names = false;
```

To check if MassPart property mp.example is a parameter by using the GetParameter method:

```
if (mp.ViewParameters().GetParameter(mp.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this mass part (*ELEMENT_MASS_PART) **Note that a carriage return is not added**. See also MassPart.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for mass part mp:

```
var key = mp.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the mass part. **Note that a carriage return is not added**. See also MassPart.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for mass part mp:

```
var cards = mp.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last element mass part in the model.

Arguments

- **Model** (Model)

Model to get last element mass part in

Returns

MassPart object (or null if there are no element mass parts in the model).

Return type

MassPart

Example

To get the last element mass part in model m:

```
var mp = MassPart.Last(m);
```

Next()

Description

Returns the next element mass part in the model.

Arguments

No arguments

Returns

MassPart object (or null if there are no more element mass parts in the model).

Return type

MassPart

Example

To get the element mass part in model m after element mass part mp:

```
var mp = mp.Next();
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a element mass part.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only element mass parts from that model can be picked. If the argument is a Flag then only element mass parts that are flagged with *limit* can be selected. If omitted, or null, any element mass parts from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

MassPart object (or null if not picked)

Return type

MassPart

Example

To pick a element mass part from model m giving the prompt 'Pick element mass part from screen':

```
var mp = MassPart.Pick('Pick element mass part from screen', m);
```

Previous()

Description

Returns the previous element mass part in the model.

Arguments

No arguments

Returns

MassPart object (or null if there are no more element mass parts in the model).

Return type

MassPart

Example

To get the element mass part in model m before element mass part mp:

```
var mp = mp.Previous();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select element mass parts using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting element mass parts

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only element mass parts from that model can be selected. If the argument is a Flag then only element mass parts that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any element mass parts can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of element mass parts selected or null if menu cancelled

Return type

Number

Example

To select element mass parts from model m, flagging those selected with flag f, giving the prompt ’Select element mass parts’:

```
MassPart.Select(f, ’Select element mass parts’, m);
```

To select element mass parts, flagging those selected with flag f but limiting selection to element mass parts flagged with flag l, giving the prompt ’Select element mass parts’:

```
MassPart.Select(f, ’Select element mass parts’, l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the element mass part.

Arguments

- **flag** (Flag)

Flag to set on the element mass part

Returns

No return value

Example

To set flag f for element mass part mp:

```
mp.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the element mass part. The element mass part will be sketched until you either call MassPart.Unsketch(), MassPart.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the element mass part is sketched. If omitted redraw is true. If you want to sketch several element mass parts and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch element mass part mp:

```
mp.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged element mass parts in the model. The element mass parts will be sketched until you either call MassPart.Unsketch(), MassPart.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged element mass parts will be sketched in

- **flag** (Flag)

Flag set on the element mass parts that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the element mass parts are sketched. If omitted redraw is true. If you want to sketch flagged element mass parts several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all element mass parts flagged with flag in model m:

```
MassPart.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of element mass parts in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing element mass parts should be counted. If false or omitted referenced but undefined element mass parts will also be included in the total.

Returns

number of element mass parts

Return type

Number

Example

To get the total number of element mass parts in model m:

```
var total = MassPart.Total(m);
```

Unblank()

Description

Unblanks the element mass part

Arguments

No arguments

Returns

No return value

Example

To unblank element mass part mp:

```
mp.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the element mass parts in the model.

Arguments

- **Model** (Model)

Model that all element mass parts will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the element mass parts in model m:

```
MassPart.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged element mass parts in the model.

Arguments

- **Model** (Model)

Model that the flagged element mass parts will be unblanked in

- **flag** (Flag)

Flag set on the element mass parts that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the element mass parts in model m flagged with f:

```
MassPart.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the element mass parts in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all element mass parts will be unset in

- **flag** (Flag)

Flag to unset on the element mass parts

Returns

No return value

Example

To unset the flag f on all the element mass parts in model m:

```
MassPart.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the element mass part.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the element mass part is unsketched. If omitted redraw is true. If you want to unsketch several element mass parts and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch element mass part mp:

```
mp.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all element mass parts.

Arguments

- **Model** (Model)

Model that all element mass parts will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the element mass parts are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all element mass parts in model m:

```
MassPart.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged element mass parts in the model.

Arguments

- **Model** (Model)

Model that all element mass parts will be unsketched in

- **flag** (Flag)

Flag set on the element mass parts that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the element mass parts are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all element mass parts flagged with flag in model m:

```
MassPart.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

MassPart object.

Return type

MassPart

Example

To check if MassPart property mp.example is a parameter by using the MassPart.GetParameter() method:

```
if (mp.ViewParameters().GetParameter(mp.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for element mass part with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The element mass part object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...

autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for element mass part mp:

```
mp.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for element mass part mp:

```
mp.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for element mass part. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for element mass part mp:

```
mp.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this element mass part.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for element mass part mp:

```
var xrefs = mp.Xrefs();
```

---

toString()

Description

Creates a string containing the mass part data in keyword format. Note that this contains the keyword header and the keyword cards. See also MassPart.Keyword() and MassPart.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for mass part mp in keyword format

```
var str = mp.toString();
```

Pretensioner class

The Pretensioner class gives you access to seatbelt pretensioner cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- ExtractColour()
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)

- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

Pretensioner properties

Name	Type	Description
colour	Colour	The colour of the pretensioner
exists (read only)	logical	true if pretensioner exists, false if referred to but not defined.
include	integer	The Include file number that the pretensioner is in.
label	integer	Pretensioner number. Also see the sbprid property which is an alternative name for this.
lmtfrc	real	Limiting force
lmtpin	real	Limiting pull-in
model (read only)	integer	The Model number that the pretensioner is in.
ptlcid	integer	Loadcurve of pull-in vs time
sbprid	integer	Pretensioner number. Also see the label property which is an alternative name for this.
sbprty	integer	Pretensioner type.
sbrid	integer	Retractor number.
sbsid1	integer	Sensor number 1
sbsid2	integer	Sensor number 2
sbsid3	integer	Sensor number 3
sbsid4	integer	Sensor number 4
time	real	Time between sensor triggering and pretensioner acting.
transparency	integer	The transparency of the pretensioner (0-100) 0% is opaque, 100% is transparent.

Detailed Description

The Pretensioner class allows you to create, modify, edit and manipulate seatbelt pretensioner cards. See the documentation below for more details.

Constructor

new Pretensioner(Model*[Model]*, sbprid*[integer]*, sbprty*[integer]*, sbrid*[integer]*, ptlcid*[integer]*, sbsid1*[integer]*, sbsid2 (optional)*[integer]*, sbsid3 (optional)*[integer]*, sbsid4 (optional)*[integer]*, time (optional)*[real]*, lmtfrc (optional)*[real]*, lmtpin (optional)*[real]*)

Description

Create a new Seatbelt Pretensioner object.

Arguments

- **Model** (Model)

Model that pretensioner will be created in

- **sbprid** (integer)

Pretensioner number.

- **sbprty** (integer)

Pretensioner type.

- **sbrid** (integer)

Retractor number.

- **ptlcid** (integer)

Loadcurve of pull-in vs time

- **sbsid1** (integer)

Sensor number 1

- **sbsid2 (optional)** (integer)

Sensor number 2

- **sbsid3 (optional)** (integer)

Sensor number 3

- **sbsid4 (optional)** (integer)

Sensor number 4

- **time (optional)** (real)

Time between sensor triggering and pretensioner acting.

- **lmtfrc (optional)** (real)

Limiting force

- **lmtpin (optional)** (real)

Limiting pull-in

Returns

Pretensioner object

Return type

Pretensioner

Example

To create a new pyrotechnic seatbelt pretensioner in model m with label 100, Retractor 10, Loading curve 20 and Sensor 30:

```
var p = new Pretensioner(m, 100, 1, 10, 20, 30);
```

Details of functions

AssociateComment(Comment[*Comment*])

Description

Associates a comment with a pretensioner.

Arguments

- **Comment** (Comment)

Comment that will be attached to the pretensioner

Returns

No return value

Example

To associate comment c to the pretensioner p:

```
p.AssociateComment(c);
```

Blank()

Description

Blanks the pretensioner

Arguments

No arguments

Returns

No return value

Example

To blank pretensioner p:

```
p.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the pretensioners in the model.

Arguments

- **Model** (Model)

Model that all pretensioners will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the pretensioners in model m:

```
Pretensioner.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged pretensioners in the model.

Arguments

- **Model** (Model)

Model that all the flagged pretensioners will be blanked in

- **flag** (Flag)

Flag set on the pretensioners that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the pretensioners in model m flagged with f:

```
Pretensioner.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the pretensioner is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if pretensioner p is blanked:

```
if (p.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse pretensioner p:

```
p.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the pretensioner.

Arguments

- **flag** (Flag)

Flag to clear on the pretensioner

Returns

No return value

Example

To clear flag f for pretensioner p:

```
p.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the pretensioner. The target include of the copied pretensioner can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

Pretensioner object

Return type

Pretensioner

Example

To copy pretensioner p into pretensioner z:

```
var z = p.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a pretensioner

Arguments

- **Model** (Model)

Model that the pretensioner will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

Pretensioner object (or null if not made).

Return type

Pretensioner

Example

To start creating a pretensioner in model m:

```
var p = Pretensioner.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a pretensioner.

Arguments

- **Comment** (Comment)

Comment that will be detached from the pretensioner

Returns

No return value

Example

To detach comment c from the pretensioner p:

```
p.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit pretensioner p:

```
p.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for pretensioner with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The pretensioner object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for pretensioner p:

```
p.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for pretensioner p:

```
p.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for pretensioner. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for pretensioner p:

```
p.Error("My custom error");
```

ExtractColour()

Description

Extracts the **actual** colour used for pretensioner.
By default in PRIMER many entities such as elements get their colour automatically from the part that they are in. PRIMER cycles through 13 default colours based on the label of the entity. In this case the pretensioner colour property will return the value Colour.PART instead of the actual colour. This method will return the actual colour which is used for drawing the pretensioner.

Arguments

No arguments

Returns

colour value (integer)

Return type

Number

Example

To return the colour used for drawing pretensioner p:

```
var colour = p.ExtractColour();
```

First(Model*[Model]*) [static]

Description

Returns the first pretensioner in the model.

Arguments

- **Model** (Model)

Model to get first pretensioner in

Returns

Pretensioner object (or null if there are no pretensioners in the model).

Return type

Pretensioner

Example

To get the first pretensioner in model m:

```
var p = Pretensioner.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free pretensioner label in the model. Also see Pretensioner.LastFreeLabel(), Pretensioner.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free pretensioner label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

Pretensioner label.

Return type

Number

Example

To get the first free pretensioner label in model m:

```
var label = Pretensioner.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the pretensioners in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all pretensioners will be flagged in

- **flag** (Flag)

Flag to set on the pretensioners

Returns

No return value

Example

To flag all of the pretensioners with flag f in model m:

```
Pretensioner.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the pretensioner is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the pretensioner

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if pretensioner p has flag f set on it:

```
if (p.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each pretensioner in the model.
Note that ForEach has been designed to make looping over pretensioners as fast as possible and so has some limitations.
Firstly, a single temporary Pretensioner object is created and on each function call it is updated with the current pretensioner data. This means that you should not try to store the Pretensioner object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new pretensioners inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all pretensioners are in

- **func** (function)

Function to call for each pretensioner

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the pretensioners in model m:

```
Pretensioner.ForEach(m, test);
function test(p)
{
// p is Pretensioner object
}
```

To call function test for all of the pretensioners in model m with optional object:

```
var data = { x:0, y:0 };
Pretensioner.ForEach(m, test, data);
function test(p, extra)
{
// p is Pretensioner object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Pretensioner objects or properties for all of the pretensioners in a model in PRIMER. If the optional property argument is not given then an array of Pretensioner objects is returned. If the property argument is given, that property value for each pretensioner is returned in the array instead of a Pretensioner object

Arguments

- **Model** (Model)

Model to get pretensioners from

- **property (optional)** (string)

Name for property to get for all pretensioners in the model

Returns

Array of Pretensioner objects or properties

Return type

Array

Example

To make an array of Pretensioner objects for all of the pretensioners in model m:

```
var a = Pretensioner.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each pretensioner in model m:

```
var a = Pretensioner.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a pretensioner.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the pretensioner p:

```
var comm_array = p.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of Pretensioner objects for all of the flagged pretensioners in a model in PRIMER If the optional property argument is not given then an array of Pretensioner objects is returned. If the property argument is given, then that property value for each pretensioner is returned in the array instead of a Pretensioner object

Arguments

- **Model** (Model)

Model to get pretensioners from

- **flag** (Flag)

Flag set on the pretensioners that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged pretensioners in the model

Returns

Array of Pretensioner objects or properties

Return type

Array

Example

To make an array of Pretensioner objects for all of the pretensioners in model m flagged with f:

```
var p = Pretensioner.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the pretensioners in model m flagged with f:

```
var a = Pretensioner.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the Pretensioner object for a pretensioner ID.

Arguments

- **Model** (Model)

Model to find the pretensioner in

- **number** (integer)

number of the pretensioner you want the Pretensioner object for

Returns

Pretensioner object (or null if pretensioner does not exist).

Return type

Pretensioner

Example

To get the Pretensioner object for pretensioner 100 in model m

```
var p = Pretensioner.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a Pretensioner property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the Pretensioner.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

pretensioner property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if Pretensioner property p.example is a parameter:

```
Options.property_parameter_names = true;
if (p.GetParameter(p.example) ) do_something...
Options.property_parameter_names = false;
```

To check if Pretensioner property p.example is a parameter by using the GetParameter method:

```
if (p.ViewParameters().GetParameter(p.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this pretensioner (*ELEMENT_SEATBELT_PRETEROMETER) **Note that a carriage return is not added**. See also Pretensioner.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for pretensioner p:

```
var key = p.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the pretensioner. **Note that a carriage return is not added**. See also Pretensioner.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for pretensioner a:

```
var cards = a.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last pretensioner in the model.

Arguments

- **Model** (Model)

Model to get last pretensioner in

Returns

Pretensioner object (or null if there are no pretensioners in the model).

Return type

Pretensioner

Example

To get the last pretensioner in model m:

```
var p = Pretensioner.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free pretensioner label in the model. Also see Pretensioner.FirstFreeLabel(), Pretensioner.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free pretensioner label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

Pretensioner label.

Return type

Number

Example

To get the last free pretensioner label in model m:

```
var label = Pretensioner.LastFreeLabel(m);
```

Next()

Description

Returns the next pretensioner in the model.

Arguments

No arguments

Returns

Pretensioner object (or null if there are no more pretensioners in the model).

Return type

Pretensioner

Example

To get the pretensioner in model m after pretensioner p:

```
var p = p.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) pretensioner label in the model. Also see Pretensioner.FirstFreeLabel(), Pretensioner.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free pretensioner label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

Pretensioner label.

Return type

Number

Example

To get the next free pretensioner label in model m:

```
var label = Pretensioner.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a pretensioner.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only pretensioners from that model can be picked. If the argument is a Flag then only pretensioners that are flagged with *limit* can be selected. If omitted, or null, any pretensioners from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

Pretensioner object (or null if not picked)

Return type

Pretensioner

Example

To pick a pretensioner from model m giving the prompt 'Pick pretensioner from screen':

```
var p = Pretensioner.Pick('Pick pretensioner from screen', m);
```

Previous()

Description

Returns the previous pretensioner in the model.

Arguments

No arguments

Returns

Pretensioner object (or null if there are no more pretensioners in the model).

Return type

Pretensioner

Example

To get the pretensioner in model m before pretensioner p:

```
var p = p.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the pretensioners in the model.

Arguments

- **Model** (Model)

Model that all pretensioners will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the pretensioners in model m, from 1000000:

```
Pretensioner.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged pretensioners in the model.

Arguments

- **Model** (Model)

Model that all the flagged pretensioners will be renumbered in

- **flag** (Flag)

Flag set on the pretensioners that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the pretensioners in model m flagged with f, from 1000000:

```
Pretensioner.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select pretensioners using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting pretensioners

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only pretensioners from that model can be selected. If the argument is a Flag then only pretensioners that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any pretensioners can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of pretensioners selected or null if menu cancelled

Return type

Number

Example

To select pretensioners from model m, flagging those selected with flag f, giving the prompt 'Select pretensioners':

```
Pretensioner.Select(f, 'Select pretensioners', m);
```

To select pretensioners, flagging those selected with flag f but limiting selection to pretensioners flagged with flag l, giving the prompt 'Select pretensioners':

```
Pretensioner.Select(f, 'Select pretensioners', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the pretensioner.

Arguments

- **flag** (Flag)

Flag to set on the pretensioner

Returns

No return value

Example

To set flag f for pretensioner p:

```
p.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the pretensioner. The pretensioner will be sketched until you either call Pretensioner.Unsketch(), Pretensioner.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the pretensioner is sketched. If omitted redraw is true. If you want to sketch several pretensioners and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch pretensioner p:

```
p.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged pretensioners in the model. The pretensioners will be sketched until you either call Pretensioner.Unsketch(), Pretensioner.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged pretensioners will be sketched in

- **flag** (Flag)

Flag set on the pretensioners that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the pretensioners are sketched. If omitted redraw is true. If you want to sketch flagged pretensioners several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all pretensioners flagged with flag in model m:

```
Pretensioner.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of pretensioners in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing pretensioners should be counted. If false or omitted referenced but undefined pretensioners will also be included in the total.

Returns

number of pretensioners

Return type

Number

Example

To get the total number of pretensioners in model m:

```
var total = Pretensioner.Total(m);
```

Unblank()

Description

Unblanks the pretensioner

Arguments

No arguments

Returns

No return value

Example

To unblank pretensioner p:

```
p.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the pretensioners in the model.

Arguments

- **Model** (Model)

Model that all pretensioners will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the pretensioners in model m:

```
Pretensioner.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged pretensioners in the model.

Arguments

- **Model** (Model)

Model that the flagged pretensioners will be unblanked in

- **flag** (Flag)

Flag set on the pretensioners that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the pretensioners in model m flagged with f:

```
Pretensioner.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the pretensioners in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all pretensioners will be unset in

- **flag** (Flag)

Flag to unset on the pretensioners

Returns

No return value

Example

To unset the flag f on all the pretensioners in model m:

```
Pretensioner.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the pretensioner.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the pretensioner is unsketched. If omitted redraw is true. If you want to unsketch several pretensioners and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch pretensioner p:

```
p.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all pretensioners.

Arguments

- **Model** (Model)

Model that all pretensioners will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the pretensioners are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all pretensioners in model m:

```
Pretensioner.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged pretensioners in the model.

Arguments

- **Model** (Model)

Model that all pretensioners will be unsketched in

- **flag** (Flag)

Flag set on the pretensioners that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the pretensioners are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all pretensioners flagged with flag in model m:

```
Pretensioner.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

Pretensioner object.

Return type

Pretensioner

Example

To check if Pretensioner property p.example is a parameter by using the Pretensioner.GetParameter() method:

```
if (p.ViewParameters().GetParameter(p.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for pretensioner with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The pretensioner object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for pretensioner p:

```
p.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for pretensioner p:

```
p.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for pretensioner. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for pretensioner p:

```
p.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this pretensioner.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for pretensioner p:

```
var xrefs = p.Xrefs();
```

---

toString()

Description

Creates a string containing the pretensioner data in keyword format. Note that this contains the keyword header and the keyword cards. See also Pretensioner.Keyword() and Pretensioner.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for pretensioner p in keyword format

```
var str = p.toString();
```

Retractor class

The Retractor class gives you access to seatbelt retractor cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- ExtractColour()
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)

- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

Retractor properties

Name	Type	Description
colour	Colour	The colour of the retractor
dsid	integer	Retractor deactivation Sensor
exists (read only)	logical	true if retractor exists, false if referred to but not defined.
flopt	integer	limiting force flage
include	integer	The Include file number that the retractor is in.
label	integer	Retractor number. Also see the sbrid property which is an alternative name for this.
lcfl	integer	Loadcurve representing an adaptive multi-level load limiter
lfed	real	Fed length
llcid	integer	Loadcurve for loading (pull-out vs force)
model (read only)	integer	The Model number that the retractor is in.
nsbi	integer	Number of elements inside the retractor
pull	real	Amount of pull out between time delay ending and retractor locking
sbid	integer	Seatbelt1D number (or Set Shell number if sbrnid is negative).
sbrid	integer	Retractor number. Also see the label property which is an alternative name for this.
sbrnid	integer	Node number (or Set Node number if negative).
shell_seatbelt (read only)	logical	true if retractor is used for shell (2D) seatbelt elements.
sid1	integer	Sensor number 1
sid2	integer	Sensor number 2
sid3	integer	Sensor number 3
sid4	integer	Sensor number 4
tdel	real	Time delay after sensor triggers
transparency	integer	The transparency of the retractor (0-100) 0% is opaque, 100% is transparent.
ulcid	integer	Loadcurve for unloading (pull-out vs force)

Detailed Description

The Retractor class allows you to create, modify, edit and manipulate seatbelt retractor cards. See the documentation below for more details.

Constructor

new Retractor(Model*[Model]*, sbrid*[integer]*, sbrnid*[integer]*, sbid*[integer]*, llcid*[integer]*, sid1*[integer]*, sid2 (optional)*[integer]*, sid3 (optional)*[integer]*, sid4 (optional)*[integer]*, tdel (optional)*[real]*, pull (optional)*[real]*, ulcid (optional)*[integer]*, lfed (optional)*[real]*, lcfl (optional)*[integer]*, flopt(optional)*[integer]*)

Description

Create a new Seatbelt Retractor object.

Arguments

- **Model** (Model)

Model that retractor will be created in

- **sbrid** (integer)

Retractor number.

- **sbrnid** (integer)

Node number (or Set Node number if negative).

- **sbid** (integer)

Seatbelt number. (or Set Shell number if sbrnid is negative)

- **llcid** (integer)

Loadcurve for loading (pull-out vs force)

- **sid1** (integer)

Sensor number 1

- **sid2 (optional)** (integer)

Sensor number 2

- **sid3 (optional)** (integer)

Sensor number 3

- **sid4 (optional)** (integer)

Sensor number 4

- **tdel (optional)** (real)

Time delay after sensor triggers.

- **pull (optional)** (real)

Amount of pull out between time delay ending and retractor locking.

- **ulcid (optional)** (integer)

Loadcurve for unloading (pull-out vs force)

- **lfed (optional)** (real)

Fed length

- **lcfl (optional)** (integer)

Loadcurve representing an adaptive multi-level load limiter

- **flopt(optional)** (integer)

limiting force flage

Returns

Retractor object

Return type

Retractor

Example

To create a new seatbelt retractor in model m with label 100, retractor Node 10, Seatbelt 20, Loading curve 30 and Sensor 40:

```
var a = new Retractor(m, 100, 10, 20, 30, 40);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a retractor.

Arguments

- **Comment** (Comment)

Comment that will be attached to the retractor

Returns

No return value

Example

To associate comment c to the retractor r:

```
r.AssociateComment(c);
```

Blank()

Description

Blanks the retractor

Arguments

No arguments

Returns

No return value

Example

To blank retractor r:

```
r.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the retractors in the model.

Arguments

- **Model** (Model)

Model that all retractors will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the retractors in model m:

```
Retractor.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged retractors in the model.

Arguments

- **Model** (Model)

Model that all the flagged retractors will be blanked in

- **flag** (Flag)

Flag set on the retractors that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the retractors in model m flagged with f:

```
Retractor.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the retractor is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if retractor r is blanked:

```
if (r.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse retractor r:

```
r.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the retractor.

Arguments

- **flag** (Flag)

Flag to clear on the retractor

Returns

No return value

Example

To clear flag f for retractor r:

```
r.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the retractor. The target include of the copied retractor can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

Retractor object

Return type

Retractor

Example

To copy retractor r into retractor z:

```
var z = r.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a retractor

Arguments

- **Model** (Model)

Model that the retractor will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

Retractor object (or null if not made).

Return type

Retractor

Example

To start creating a retractor in model m:

```
var r = Retractor.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a retractor.

Arguments

- **Comment** (Comment)

Comment that will be detached from the retractor

Returns

No return value

Example

To detach comment c from the retractor r:

```
r.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit retractor r:

```
r.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for retractor with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The retractor object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for retractor r:

```
r.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for retractor r:

```
r.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for retractor. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for retractor r:

```
r.Error("My custom error");
```

ExtractColour()

Description

Extracts the **actual** colour used for retractor.
By default in PRIMER many entities such as elements get their colour automatically from the part that they are in. PRIMER cycles through 13 default colours based on the label of the entity. In this case the retractor colour property will return the value Colour.PART instead of the actual colour. This method will return the actual colour which is used for drawing the retractor.

Arguments

No arguments

Returns

colour value (integer)

Return type

Number

Example

To return the colour used for drawing retractor r:

```
var colour = r.ExtractColour();
```

First(Model*[Model]*) [static]

Description

Returns the first retractor in the model.

Arguments

- **Model** (Model)

Model to get first retractor in

Returns

Retractor object (or null if there are no retractors in the model).

Return type

Retractor

Example

To get the first retractor in model m:

```
var r = Retractor.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free retractor label in the model. Also see Retractor.LastFreeLabel(), Retractor.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free retractor label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

Retractor label.

Return type

Number

Example

To get the first free retractor label in model m:

```
var label = Retractor.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the retractors in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all retractors will be flagged in

- **flag** (Flag)

Flag to set on the retractors

Returns

No return value

Example

To flag all of the retractors with flag f in model m:

```
Retractor.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the retractor is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the retractor

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if retractor r has flag f set on it:

```
if (r.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each retractor in the model.
Note that ForEach has been designed to make looping over retractors as fast as possible and so has some limitations.
Firstly, a single temporary Retractor object is created and on each function call it is updated with the current retractor data. This means that you should not try to store the Retractor object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new retractors inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all retractors are in

- **func** (function)

Function to call for each retractor

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the retractors in model m:

```
Retractor.ForEach(m, test);
function test(r)
{
// r is Retractor object
}
```

To call function test for all of the retractors in model m with optional object:

```
var data = { x:0, y:0 };
Retractor.ForEach(m, test, data);
function test(r, extra)
{
// r is Retractor object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Retractor objects or properties for all of the retractors in a model in PRIMER. If the optional property argument is not given then an array of Retractor objects is returned. If the property argument is given, that property value for each retractor is returned in the array instead of a Retractor object

Arguments

- **Model** (Model)

Model to get retractors from

- **property (optional)** (string)

Name for property to get for all retractors in the model

Returns

Array of Retractor objects or properties

Return type

Array

Example

To make an array of Retractor objects for all of the retractors in model m:

```
var a = Retractor.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each retractor in model m:

```
var a = Retractor.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a retractor.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the retractor r:

```
var comm_array = r.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of Retractor objects for all of the flagged retractors in a model in PRIMER If the optional property argument is not given then an array of Retractor objects is returned. If the property argument is given, then that property value for each retractor is returned in the array instead of a Retractor object

Arguments

- **Model** (Model)

Model to get retractors from

- **flag** (Flag)

Flag set on the retractors that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged retractors in the model

Returns

Array of Retractor objects or properties

Return type

Array

Example

To make an array of Retractor objects for all of the retractors in model m flagged with f:

```
var r = Retractor.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the retractors in model m flagged with f:

```
var a = Retractor.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the Retractor object for a retractor ID.

Arguments

- **Model** (Model)

Model to find the retractor in

- **number** (integer)

number of the retractor you want the Retractor object for

Returns

Retractor object (or null if retractor does not exist).

Return type

Retractor

Example

To get the Retractor object for retractor 100 in model m

```
var r = Retractor.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a Retractor property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the Retractor.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

retractor property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if Retractor property r.example is a parameter:

```
Options.property_parameter_names = true;
if (r.GetParameter(r.example) ) do_something...
Options.property_parameter_names = false;
```

To check if Retractor property r.example is a parameter by using the GetParameter method:

```
if (r.ViewParameters().GetParameter(r.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this retractor (*ELEMENT_SEATBELT_RETREROMETER) **Note that a carriage return is not added**. See also Retractor.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for retractor r:

```
var key = r.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the retractor. **Note that a carriage return is not added**. See also Retractor.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for retractor r:

```
var cards = r.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last retractor in the model.

Arguments

- **Model** (Model)

Model to get last retractor in

Returns

Retractor object (or null if there are no retractors in the model).

Return type

Retractor

Example

To get the last retractor in model m:

```
var r = Retractor.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free retractor label in the model. Also see Retractor.FirstFreeLabel(), Retractor.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free retractor label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

Retractor label.

Return type

Number

Example

To get the last free retractor label in model m:

```
var label = Retractor.LastFreeLabel(m);
```

Next()

Description

Returns the next retractor in the model.

Arguments

No arguments

Returns

Retractor object (or null if there are no more retractors in the model).

Return type

Retractor

Example

To get the retractor in model m after retractor r:

```
var r = r.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) retractor label in the model. Also see Retractor.FirstFreeLabel(), Retractor.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free retractor label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

Retractor label.

Return type

Number

Example

To get the next free retractor label in model m:

```
var label = Retractor.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a retractor.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only retractors from that model can be picked. If the argument is a Flag then only retractors that are flagged with *limit* can be selected. If omitted, or null, any retractors from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

Retractor object (or null if not picked)

Return type

Retractor

Example

To pick a retractor from model m giving the prompt 'Pick retractor from screen':

```
var r = Retractor.Pick('Pick retractor from screen', m);
```

Previous()

Description

Returns the previous retractor in the model.

Arguments

No arguments

Returns

Retractor object (or null if there are no more retractors in the model).

Return type

Retractor

Example

To get the retractor in model m before retractor r:

```
var r = r.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the retractors in the model.

Arguments

- **Model** (Model)

Model that all retractors will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the retractors in model m, from 1000000:

```
Retractor.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged retractors in the model.

Arguments

- **Model** (Model)

Model that all the flagged retractors will be renumbered in

- **flag** (Flag)

Flag set on the retractors that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the retractors in model m flagged with f, from 1000000:

```
Retractor.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select retractors using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting retractors

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only retractors from that model can be selected. If the argument is a Flag then only retractors that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any retractors can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of retractors selected or null if menu cancelled

Return type

Number

Example

To select retractors from model m, flagging those selected with flag f, giving the prompt 'Select retractors':

```
Retractor.Select(f, 'Select retractors', m);
```

To select retractors, flagging those selected with flag f but limiting selection to retractors flagged with flag l, giving the prompt 'Select retractors':

```
Retractor.Select(f, 'Select retractors', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the retractor.

Arguments

- **flag** (Flag)

Flag to set on the retractor

Returns

No return value

Example

To set flag f for retractor r:

```
r.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the retractor. The retractor will be sketched until you either call Retractor.Unsketch(), Retractor.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the retractor is sketched. If omitted redraw is true. If you want to sketch several retractors and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch retractor r:

```
r.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged retractors in the model. The retractors will be sketched until you either call Retractor.Unsketch(), Retractor.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged retractors will be sketched in

- **flag** (Flag)

Flag set on the retractors that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the retractors are sketched. If omitted redraw is true. If you want to sketch flagged retractors several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all retractors flagged with flag in model m:

```
Retractor.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of retractors in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing retractors should be counted. If false or omitted referenced but undefined retractors will also be included in the total.

Returns

number of retractors

Return type

Number

Example

To get the total number of retractors in model m:

```
var total = Retractor.Total(m);
```

Unblank()

Description

Unblanks the retractor

Arguments

No arguments

Returns

No return value

Example

To unblank retractor r:

```
r.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the retractors in the model.

Arguments

- **Model** (Model)

Model that all retractors will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the retractors in model m:

```
Retractor.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged retractors in the model.

Arguments

- **Model** (Model)

Model that the flagged retractors will be unblanked in

- **flag** (Flag)

Flag set on the retractors that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the retractors in model m flagged with f:

```
Retractor.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the retractors in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all retractors will be unset in

- **flag** (Flag)

Flag to unset on the retractors

Returns

No return value

Example

To unset the flag f on all the retractors in model m:

```
Retractor.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the retractor.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the retractor is unsketched. If omitted redraw is true. If you want to unsketch several retractors and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch retractor r:

```
r.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all retractors.

Arguments

- **Model** (Model)

Model that all retractors will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the retractors are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all retractors in model m:

```
Retractor.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged retractors in the model.

Arguments

- **Model** (Model)

Model that all retractors will be unsketched in

- **flag** (Flag)

Flag set on the retractors that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the retractors are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all retractors flagged with flag in model m:

```
Retractor.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

Retractor object.

Return type

Retractor

Example

To check if Retractor property r.example is a parameter by using the Retractor.GetParameter() method:

```
if (r.ViewParameters().GetParameter(r.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for retractor with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The retractor object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for retractor r:

```
r.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for retractor r:

```
r.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for retractor. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for retractor r:

```
r.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this retractor.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for retractor r:

```
var xrefs = r.Xrefs();
```

---

toString()

Description

Creates a string containing the retractor data in keyword format. Note that this contains the keyword header and the keyword cards. See also Retractor.Keyword() and Retractor.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for retractor r in keyword format

```
var str = r.toString();
```

Seatbelt1D class

The Seatbelt1D class gives you access to 2 noded (1D) element seatbelt cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- ExtractColour()
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Timestep()
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()

- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

Seatbelt1D properties

Name	Type	Description
colour	Colour	The colour of the seatbelt
eid	integer	Seatbelt1D number. Also see the label property which is an alternative name for this.
exists (read only)	logical	true if seatbelt exists, false if referred to but not defined.
include	integer	The Include file number that the seatbelt is in.
label	integer	Seatbelt1D number. Also see the eid property which is an alternative name for this.
model (read only)	integer	The Model number that the seatbelt is in.
n1	integer	Node 1 ID
n2	integer	Node 2 ID
pid	integer	Part ID
sbrid	integer	Retractor ID
slen	real	Initial slack length
transparency	integer	The transparency of the seatbelt (0-100) 0% is opaque, 100% is transparent.

Detailed Description

The Seatbelt1D class allows you to create, modify, edit and manipulate 2 noded (1D) element seatbelt cards. See the documentation below for more details.

Constructor

new Seatbelt1D(Model*[Model]*, eid*[integer]*, pid*[integer]*, n1*[integer]*, n2*[integer]*)

Description

Create a new Seatbelt1D object.

Arguments

- **Model** (Model)

Model that seatbelt will be created in

- **eid** (integer)

Seatbelt ID.

- **pid** (integer)

Part number.

- **n1** (integer)

Node 1 ID

- **n2** (integer)

Node 2 ID

Returns

Seatbelt1D object

Return type

Seatbelt1D

Example

To create a new 2 noded element seatbelt in model m with label 100, part 10 and nodes 20, 21:

```
var a = new Seatbelt1D(m, 100, 10, 20, 21);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a seatbelt.

Arguments

- **Comment** (Comment)

Comment that will be attached to the seatbelt

Returns

No return value

Example

To associate comment c to the seatbelt s:

```
s.AssociateComment(c);
```

Blank()

Description

Blanks the seatbelt

Arguments

No arguments

Returns

No return value

Example

To blank seatbelt s:

```
s.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the seatbelts in the model.

Arguments

- **Model** (Model)

Model that all seatbelts will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the seatbelts in model m:

```
Seatbelt1D.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged seatbelts in the model.

Arguments

- **Model** (Model)

Model that all the flagged seatbelts will be blanked in

- **flag** (Flag)

Flag set on the seatbelts that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the seatbelts in model m flagged with f:

```
Seatbelt1D.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the seatbelt is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if seatbelt s is blanked:

```
if (s.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse seatbelt s:

```
s.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the seatbelt.

Arguments

- **flag** (Flag)

Flag to clear on the seatbelt

Returns

No return value

Example

To clear flag f for seatbelt s:

```
s.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the seatbelt. The target include of the copied seatbelt can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

Seatbelt1D object

Return type

Seatbelt1D

Example

To copy seatbelt s into seatbelt z:

```
var z = s.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a seatbelt

Arguments

- **Model** (Model)

Model that the seatbelt will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

Seatbelt1D object (or null if not made).

Return type

Seatbelt1D

Example

To start creating a seatbelt in model m:

```
var s = Seatbelt1D.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a seatbelt.

Arguments

- **Comment** (Comment)

Comment that will be detached from the seatbelt

Returns

No return value

Example

To detach comment c from the seatbelt s:

```
s.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit seatbelt s:

```
s.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for seatbelt with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The seatbelt object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for seatbelt s:

```
s.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for seatbelt s:

```
s.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for seatbelt. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for seatbelt s:

```
s.Error("My custom error");
```

ExtractColour()

Description

Extracts the **actual** colour used for seatbelt.
By default in PRIMER many entities such as elements get their colour automatically from the part that they are in. PRIMER cycles through 13 default colours based on the label of the entity. In this case the seatbelt colour property will return the value Colour.PART instead of the actual colour. This method will return the actual colour which is used for drawing the seatbelt.

Arguments

No arguments

Returns

colour value (integer)

Return type

Number

Example

To return the colour used for drawing seatbelt s:

```
var colour = s.ExtractColour();
```

First(Model*[Model]*) [static]

Description

Returns the first seatbelt in the model.

Arguments

- **Model** (Model)

Model to get first seatbelt in

Returns

Seatbelt1D object (or null if there are no seatbelts in the model).

Return type

Seatbelt1D

Example

To get the first seatbelt in model m:

```
var s = Seatbelt1D.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free seatbelt label in the model. Also see Seatbelt1D.LastFreeLabel(), Seatbelt1D.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free seatbelt label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

Seatbelt1D label.

Return type

Number

Example

To get the first free seatbelt label in model m:

```
var label = Seatbelt1D.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the seatbelts in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all seatbelts will be flagged in

- **flag** (Flag)

Flag to set on the seatbelts

Returns

No return value

Example

To flag all of the seatbelts with flag f in model m:

```
Seatbelt1D.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the seatbelt is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the seatbelt

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if seatbelt s has flag f set on it:

```
if (s.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each seatbelt in the model.
Note that ForEach has been designed to make looping over seatbelts as fast as possible and so has some limitations.
Firstly, a single temporary Seatbelt1D object is created and on each function call it is updated with the current seatbelt data. This means that you should not try to store the Seatbelt1D object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new seatbelts inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all seatbelts are in

- **func** (function)

Function to call for each seatbelt

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the seatbelts in model m:

```
Seatbelt1D.ForEach(m, test);
function test(s)
{
// s is Seatbelt1D object
}
```

To call function test for all of the seatbelts in model m with optional object:

```
var data = { x:0, y:0 };
Seatbelt1D.ForEach(m, test, data);
function test(s, extra)
{
// s is Seatbelt1D object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Seatbelt1D objects or properties for all of the seatbelts in a model in PRIMER. If the optional property argument is not given then an array of Seatbelt1D objects is returned. If the property argument is given, that property value for each seatbelt is returned in the array instead of a Seatbelt1D object

Arguments

- **Model** (Model)

Model to get seatbelts from

- **property (optional)** (string)

Name for property to get for all seatbelts in the model

Returns

Array of Seatbelt1D objects or properties

Return type

Array

Example

To make an array of Seatbelt1D objects for all of the seatbelts in model m:

```
var a = Seatbelt1D.GetAll(m);
```

To return an array containing the value of property ’foo’ (for example ’x’ for a node) for each seatbelt in model m:

```
var a = Seatbelt1D.GetAll(m, ’foo’);
```

GetComments()

Description

Extracts the comments associated to a seatbelt.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the seatbelt s:

```
var comm_array = s.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of Seatbelt1D objects for all of the flagged seatbelts in a model in PRIMER If the optional property argument is not given then an array of Seatbelt1D objects is returned. If the property argument is given, then that property value for each seatbelt is returned in the array instead of a Seatbelt1D object

Arguments

- **Model** (Model)

Model to get seatbelts from

- **flag** (Flag)

Flag set on the seatbelts that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged seatbelts in the model

Returns

Array of Seatbelt1D objects or properties

Return type

Array

Example

To make an array of Seatbelt1D objects for all of the seatbelts in model m flagged with f:

```
var s = Seatbelt1D.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the seatbelts in model m flagged with f:

```
var a = Seatbelt1D.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the Seatbelt1D object for a seatbelt ID.

Arguments

- **Model** (Model)

Model to find the seatbelt in

- **number** (integer)

number of the seatbelt you want the Seatbelt1D object for

Returns

Seatbelt1D object (or null if seatbelt does not exist).

Return type

Seatbelt1D

Example

To get the Seatbelt1D object for seatbelt 100 in model m

```
var s = Seatbelt1D.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a Seatbelt1D property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the Seatbelt1D.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

seatbelt property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if Seatbelt1D property s.example is a parameter:

```
Options.property_parameter_names = true;
if (s.GetParameter(s.example) ) do_something...
Options.property_parameter_names = false;
```

To check if Seatbelt1D property s.example is a parameter by using the GetParameter method:

```
if (s.ViewParameters().GetParameter(s.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this seatbelt (*ELEMENT_SEATBELT) **Note that a carriage return is not added**. See also Seatbelt1D.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for seatbelt s:

```
var key = s.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the seatbelt. **Note that a carriage return is not added**. See also Seatbelt1D.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for seatbelt s:

```
var cards = s.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last seatbelt in the model.

Arguments

- **Model** (Model)

Model to get last seatbelt in

Returns

Seatbelt1D object (or null if there are no seatbelts in the model).

Return type

Seatbelt1D

Example

To get the last seatbelt in model m:

```
var s = Seatbelt1D.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free seatbelt label in the model. Also see Seatbelt1D.FirstFreeLabel(), Seatbelt1D.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free seatbelt label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

Seatbelt1D label.

Return type

Number

Example

To get the last free seatbelt label in model m:

```
var label = Seatbelt1D.LastFreeLabel(m);
```

Next()

Description

Returns the next seatbelt in the model.

Arguments

No arguments

Returns

Seatbelt1D object (or null if there are no more seatbelts in the model).

Return type

Seatbelt1D

Example

To get the seatbelt in model m after seatbelt s:

```
var s = s.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) seatbelt label in the model. Also see Seatbelt1D.FirstFreeLabel(), Seatbelt1D.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free seatbelt label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

Seatbelt1D label.

Return type

Number

Example

To get the next free seatbelt label in model m:

```
var label = Seatbelt1D.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a seatbelt.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only seatbelts from that model can be picked. If the argument is a Flag then only seatbelts that are flagged with *limit* can be selected. If omitted, or null, any seatbelts from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

Seatbelt1D object (or null if not picked)

Return type

Seatbelt1D

Example

To pick a seatbelt from model m giving the prompt 'Pick seatbelt from screen':

```
var s = Seatbelt1D.Pick('Pick seatbelt from screen', m);
```

Previous()

Description

Returns the previous seatbelt in the model.

Arguments

No arguments

Returns

Seatbelt1D object (or null if there are no more seatbelts in the model).

Return type

Seatbelt1D

Example

To get the seatbelt in model m before seatbelt s:

```
var s = s.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the seatbelts in the model.

Arguments

- **Model** (Model)

Model that all seatbelts will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the seatbelts in model m, from 1000000:

```
Seatbelt1D.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged seatbelts in the model.

Arguments

- **Model** (Model)

Model that all the flagged seatbelts will be renumbered in

- **flag** (Flag)

Flag set on the seatbelts that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the seatbelts in model m flagged with f, from 1000000:

```
Seatbelt1D.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select seatbelts using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting seatbelts

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only seatbelts from that model can be selected. If the argument is a Flag then only seatbelts that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any seatbelts can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of seatbelts selected or null if menu cancelled

Return type

Number

Example

To select seatbelts from model m, flagging those selected with flag f, giving the prompt 'Select seatbelts':

```
Seatbelt1D.Select(f, 'Select seatbelts', m);
```

To select seatbelts, flagging those selected with flag f but limiting selection to seatbelts flagged with flag l, giving the prompt 'Select seatbelts':

```
Seatbelt1D.Select(f, 'Select seatbelts', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the seatbelt.

Arguments

- **flag** (Flag)

Flag to set on the seatbelt

Returns

No return value

Example

To set flag f for seatbelt s:

```
s.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the seatbelt. The seatbelt will be sketched until you either call Seatbelt1D.Unsketch(), Seatbelt1D.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the seatbelt is sketched. If omitted redraw is true. If you want to sketch several seatbelts and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch seatbelt s:

```
s.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged seatbelts in the model. The seatbelts will be sketched until you either call Seatbelt1D.Unsketch(), Seatbelt1D.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged seatbelts will be sketched in

- **flag** (Flag)

Flag set on the seatbelts that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the seatbelts are sketched. If omitted redraw is true. If you want to sketch flagged seatbelts several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all seatbelts flagged with flag in model m:

```
Seatbelt1D.SketchFlagged(m, flag);
```

Timestep()

Description

Calculates the timestep for the seatbelt

Arguments

No arguments

Returns

real

Return type

Number

Example

To calculate the timestep for seatbelt s:

```
var timestep = s.Timestep();
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of seatbelts in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing seatbelts should be counted. If false or omitted referenced but undefined seatbelts will also be included in the total.

Returns

number of seatbelts

Return type

Number

Example

To get the total number of seatbelts in model m:

```
var total = Seatbelt1D.Total(m);
```

Unblank()

Description

Unblanks the seatbelt

Arguments

No arguments

Returns

No return value

Example

To unblank seatbelt s:

```
s.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the seatbelts in the model.

Arguments

- **Model** (Model)

Model that all seatbelts will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the seatbelts in model m:

```
Seatbelt1D.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged seatbelts in the model.

Arguments

- **Model** (Model)

Model that the flagged seatbelts will be unblanked in

- **flag** (Flag)

Flag set on the seatbelts that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the seatbelts in model m flagged with f:

```
Seatbelt1D.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the seatbelts in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all seatbelts will be unset in

- **flag** (Flag)

Flag to unset on the seatbelts

Returns

No return value

Example

To unset the flag f on all the seatbelts in model m:

```
Seatbelt1D.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the seatbelt.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the seatbelt is unsketched. If omitted redraw is true. If you want to unsketch several seatbelts and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch seatbelt s:

```
s.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all seatbelts.

Arguments

- **Model** (Model)

Model that all seatbelts will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the seatbelts are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all seatbelts in model m:

```
Seatbelt1D.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged seatbelts in the model.

Arguments

- **Model** (Model)

Model that all seatbelts will be unsketched in

- **flag** (Flag)

Flag set on the seatbelts that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the seatbelts are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all seatbelts flagged with flag in model m:

```
Seatbelt1D.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

Seatbelt1D object.

Return type

Seatbelt1D

Example

To check if Seatbelt1D property s.example is a parameter by using the Seatbelt1D.GetParameter() method:

```
if (s.ViewParameters().GetParameter(s.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for seatbelt with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The seatbelt object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for seatbelt s:

```
s.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for seatbelt s:

```
s.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for seatbelt. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for seatbelt s:

```
s.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this seatbelt.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for seatbelt s:

```
var xrefs = s.Xrefs();
```

---

toString()

Description

Creates a string containing the seatbelt data in keyword format. Note that this contains the keyword header and the keyword cards. See also Seatbelt1D.Keyword() and Seatbelt1D.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for seatbelt s in keyword format

```
var str = s.toString();
```

Seatbelt2D class

The Seatbelt2D class gives you access to 4 noded (2D) element seatbelt cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- ExtractColour()
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()

- toString()

Seatbelt2D properties

Name	Type	Description
colour	Colour	The colour of the seatbelt
eid	integer	Seatbelt2D number. Also see the label property which is an alternative name for this.
exists (read only)	logical	true if seatbelt exists, false if referred to but not defined.
include	integer	The Include file number that the seatbelt is in.
label	integer	Seatbelt2D number. Also see the eid property which is an alternative name for this.
model (read only)	integer	The Model number that the seatbelt is in.
n1	integer	Node 1 ID
n2	integer	Node 2 ID
n3	integer	Node 3 ID
n4	integer	Node 4 ID
pid	integer	Part ID
sbrid	integer	Retractor ID
slen	real	Initial slack length
transparency	integer	The transparency of the seatbelt (0-100) 0% is opaque, 100% is transparent.

Detailed Description

The Seatbelt2D class allows you to create, modify, edit and manipulate 4 noded element seatbelt cards. See the documentation below for more details.

Constructor

new Seatbelt2D(Model*[Model]*, eid*[integer]*, pid*[integer]*, n1*[integer]*, n2*[integer]*, n3*[integer]*, n4*[integer]*)

Description

Create a new Seatbelt2D object.

Arguments

- **Model** (Model)

Model that seatbelt will be created in

- **eid** (integer)

Seatbelt ID.

- **pid** (integer)

Part number.

- **n1** (integer)

Node 1 ID

- **n2** (integer)

Node 2 ID

- **n3** (integer)

Node 3 ID

- **n4** (integer)

Node 4 ID

Returns

Seatbelt2D object

Return type

Seatbelt2D

Example

To create a new 4 noded element seatbelt in model m with label 100, part 10 and nodes 20, 21, 22, 23:

```
var a = new Seatbelt2D(m, 100, 10, 20, 21, 22, 23);
```

Details of functions

AssociateComment(Comment[*Comment*])

Description

Associates a comment with a seatbelt.

Arguments

- **Comment** (Comment)

Comment that will be attached to the seatbelt

Returns

No return value

Example

To associate comment c to the seatbelt s:

```
s.AssociateComment(c);
```

Blank()

Description

Blanks the seatbelt

Arguments

No arguments

Returns

No return value

Example

To blank seatbelt s:

```
s.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the seatbelts in the model.

Arguments

- **Model** (Model)

Model that all seatbelts will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the seatbelts in model m:

```
Seatbelt2D.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged seatbelts in the model.

Arguments

- **Model** (Model)

Model that all the flagged seatbelts will be blanked in

- **flag** (Flag)

Flag set on the seatbelts that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the seatbelts in model m flagged with f:

```
Seatbelt2D.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the seatbelt is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if seatbelt s is blanked:

```
if (s.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse seatbelt s:

```
s.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the seatbelt.

Arguments

- **flag** (Flag)

Flag to clear on the seatbelt

Returns

No return value

Example

To clear flag f for seatbelt s:

```
s.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the seatbelt. The target include of the copied seatbelt can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

Seatbelt2D object

Return type

Seatbelt2D

Example

To copy seatbelt s into seatbelt z:

```
var z = s.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a seatbelt

Arguments

- **Model** (Model)

Model that the seatbelt will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

Seatbelt2D object (or null if not made).

Return type

Seatbelt2D

Example

To start creating a seatbelt in model m:

```
var s = Seatbelt2D.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a seatbelt.

Arguments

- **Comment** (Comment)

Comment that will be detached from the seatbelt

Returns

No return value

Example

To detach comment c from the seatbelt s:

```
s.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit seatbelt s:

```
s.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for seatbelt with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The seatbelt object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for seatbelt s:

```
s.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for seatbelt s:

```
s.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for seatbelt. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for seatbelt s:

```
s.Error("My custom error");
```

ExtractColour()

Description

Extracts the **actual** colour used for seatbelt.
By default in PRIMER many entities such as elements get their colour automatically from the part that they are in. PRIMER cycles through 13 default colours based on the label of the entity. In this case the seatbelt colour property will return the value Colour.PART instead of the actual colour. This method will return the actual colour which is used for drawing the seatbelt.

Arguments

No arguments

Returns

colour value (integer)

Return type

Number

Example

To return the colour used for drawing seatbelt s:

```
var colour = s.ExtractColour();
```

First(Model*[Model]*) [static]

Description

Returns the first seatbelt in the model.

Arguments

- **Model** (Model)

Model to get first seatbelt in

Returns

Seatbelt2D object (or null if there are no seatbelts in the model).

Return type

Seatbelt2D

Example

To get the first seatbelt in model m:

```
var s = Seatbelt2D.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free seatbelt label in the model. Also see Seatbelt2D.LastFreeLabel(), Seatbelt2D.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free seatbelt label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

Seatbelt2D label.

Return type

Number

Example

To get the first free seatbelt label in model m:

```
var label = Seatbelt2D.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the seatbelts in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all seatbelts will be flagged in

- **flag** (Flag)

Flag to set on the seatbelts

Returns

No return value

Example

To flag all of the seatbelts with flag f in model m:

```
Seatbelt2D.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the seatbelt is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the seatbelt

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if seatbelt s has flag f set on it:

```
if (s.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each seatbelt in the model.
Note that ForEach has been designed to make looping over seatbelts as fast as possible and so has some limitations.
Firstly, a single temporary Seatbelt2D object is created and on each function call it is updated with the current seatbelt data. This means that you should not try to store the Seatbelt2D object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new seatbelts inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all seatbelts are in

- **func** (function)

Function to call for each seatbelt

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the seatbelts in model m:

```
Seatbelt2D.ForEach(m, test);
function test(s)
{
// s is Seatbelt2D object
}
```

To call function test for all of the seatbelts in model m with optional object:

```
var data = { x:0, y:0 };
Seatbelt2D.ForEach(m, test, data);
function test(s, extra)
{
// s is Seatbelt2D object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Seatbelt2D objects or properties for all of the seatbelts in a model in PRIMER. If the optional property argument is not given then an array of Seatbelt2D objects is returned. If the property argument is given, that property value for each seatbelt is returned in the array instead of a Seatbelt2D object

Arguments

- **Model** (Model)

Model to get seatbelts from

- **property (optional)** (string)

Name for property to get for all seatbelts in the model

Returns

Array of Seatbelt2D objects or properties

Return type

Array

Example

To make an array of Seatbelt2D objects for all of the seatbelts in model m:

```
var a = Seatbelt2D.GetAll(m);
```

To return an array containing the value of property ’foo’ (for example ’x’ for a node) for each seatbelt in model m:

```
var a = Seatbelt2D.GetAll(m, ’foo’);
```

GetComments()

Description

Extracts the comments associated to a seatbelt.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the seatbelt s:

```
var comm_array = s.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of Seatbelt2D objects for all of the flagged seatbelts in a model in PRIMER If the optional property argument is not given then an array of Seatbelt2D objects is returned. If the property argument is given, then that property value for each seatbelt is returned in the array instead of a Seatbelt2D object

Arguments

- **Model** (Model)

Model to get seatbelts from

- **flag** (Flag)

Flag set on the seatbelts that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged seatbelts in the model

Returns

Array of Seatbelt2D objects or properties

Return type

Array

Example

To make an array of Seatbelt2D objects for all of the seatbelts in model m flagged with f:

```
var s = Seatbelt2D.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the seatbelts in model m flagged with f:

```
var a = Seatbelt2D.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the Seatbelt2D object for a seatbelt ID.

Arguments

- **Model** (Model)

Model to find the seatbelt in

- **number** (integer)

number of the seatbelt you want the Seatbelt2D object for

Returns

Seatbelt2D object (or null if seatbelt does not exist).

Return type

Seatbelt2D

Example

To get the Seatbelt2D object for seatbelt 100 in model m

```
var s = Seatbelt2D.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a Seatbelt2D property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the Seatbelt2D.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

seatbelt property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if Seatbelt2D property s.example is a parameter:

```
Options.property_parameter_names = true;
if (s.GetParameter(s.example) ) do_something...
Options.property_parameter_names = false;
```

To check if Seatbelt2D property s.example is a parameter by using the GetParameter method:

```
if (s.ViewParameters().GetParameter(s.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this seatbelt (*ELEMENT_SEATBELT) **Note that a carriage return is not added**. See also Seatbelt2D.KeywordCards()

Arguments

No arguments

Returns
string containing the keyword.

Return type
String

Example
To get the keyword for seatbelt s:
```
var key = s.Keyword();
```

KeywordCards()

Description
Returns the keyword cards for the seatbelt. **Note that a carriage return is not added**. See also Seatbelt2D.Keyword()

Arguments
No arguments

Returns
string containing the cards.

Return type
String

Example
To get the cards for seatbelt s:
```
var cards = s.KeywordCards();
```

Last(Model[*Model*]) [static]

Description
Returns the last seatbelt in the model.

Arguments
- **Model** (Model)

Model to get last seatbelt in

Returns
Seatbelt2D object (or null if there are no seatbelts in the model).

Return type
Seatbelt2D

Example
To get the last seatbelt in model m:
```
var s = Seatbelt2D.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free seatbelt label in the model. Also see Seatbelt2D.FirstFreeLabel(), Seatbelt2D.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free seatbelt label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

Seatbelt2D label.

Return type

Number

Example

To get the last free seatbelt label in model m:

```
var label = Seatbelt2D.LastFreeLabel(m);
```

Next()

Description

Returns the next seatbelt in the model.

Arguments

No arguments

Returns

Seatbelt2D object (or null if there are no more seatbelts in the model).

Return type

Seatbelt2D

Example

To get the seatbelt in model m after seatbelt s:

```
var s = s.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) seatbelt label in the model. Also see Seatbelt2D.FirstFreeLabel(), Seatbelt2D.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free seatbelt label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

Seatbelt2D label.

Return type

Number

Example

To get the next free seatbelt label in model m:

```
var label = Seatbelt2D.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a seatbelt.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only seatbelts from that model can be picked. If the argument is a Flag then only seatbelts that are flagged with *limit* can be selected. If omitted, or null, any seatbelts from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

Seatbelt2D object (or null if not picked)

Return type

Seatbelt2D

Example

To pick a seatbelt from model m giving the prompt 'Pick seatbelt from screen':

```
var s = Seatbelt2D.Pick('Pick seatbelt from screen', m);
```

Previous()

Description

Returns the previous seatbelt in the model.

Arguments

No arguments

Returns

Seatbelt2D object (or null if there are no more seatbelts in the model).

Return type

Seatbelt2D

Example

To get the seatbelt in model m before seatbelt s:

```
var s = s.Previous();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select seatbelts using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting seatbelts

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only seatbelts from that model can be selected. If the argument is a Flag then only seatbelts that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any seatbelts can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of seatbelts selected or null if menu cancelled

Return type

Number

Example

To select seatbelts from model m, flagging those selected with flag f, giving the prompt 'Select seatbelts':

```
Seatbelt2D.Select(f, 'Select seatbelts', m);
```

To select seatbelts, flagging those selected with flag f but limiting selection to seatbelts flagged with flag l, giving the prompt 'Select seatbelts':

```
Seatbelt2D.Select(f, 'Select seatbelts', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the seatbelt.

Arguments

- **flag** (Flag)

Flag to set on the seatbelt

Returns

No return value

Example

To set flag f for seatbelt s:

```
s.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the seatbelt. The seatbelt will be sketched until you either call Seatbelt2D.Unsketch(), Seatbelt2D.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the seatbelt is sketched. If omitted redraw is true. If you want to sketch several seatbelts and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch seatbelt s:

```
s.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged seatbelts in the model. The seatbelts will be sketched until you either call Seatbelt2D.Unsketch(), Seatbelt2D.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged seatbelts will be sketched in

- **flag** (Flag)

Flag set on the seatbelts that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the seatbelts are sketched. If omitted redraw is true. If you want to sketch flagged seatbelts several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all seatbelts flagged with flag in model m:

```
Seatbelt2D.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of seatbelts in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing seatbelts should be counted. If false or omitted referenced but undefined seatbelts will also be included in the total.

Returns

number of seatbelts

Return type

Number

Example

To get the total number of seatbelts in model m:

```
var total = Seatbelt2D.Total(m);
```

Unblank()

Description

Unblanks the seatbelt

Arguments

No arguments

Returns

No return value

Example

To unblank seatbelt s:

```
s.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the seatbelts in the model.

Arguments

- **Model** (Model)

Model that all seatbelts will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the seatbelts in model m:

```
Seatbelt2D.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged seatbelts in the model.

Arguments

- **Model** (Model)

Model that the flagged seatbelts will be unblanked in

- **flag** (Flag)

Flag set on the seatbelts that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the seatbelts in model m flagged with f:

```
Seatbelt2D.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the seatbelts in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all seatbelts will be unset in

- **flag** (Flag)

Flag to unset on the seatbelts

Returns

No return value

Example

To unset the flag f on all the seatbelts in model m:

```
Seatbelt2D.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the seatbelt.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the seatbelt is unsketched. If omitted redraw is true. If you want to unsketch several seatbelts and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch seatbelt s:

```
s.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all seatbelts.

Arguments

- **Model** (Model)

Model that all seatbelts will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the seatbelts are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all seatbelts in model m:

```
Seatbelt2D.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged seatbelts in the model.

Arguments

- **Model** (Model)

Model that all seatbelts will be unsketched in

- **flag** (Flag)

Flag set on the seatbelts that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the seatbelts are unsketched. If omitted redraw is true. If you want to unsketch

several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all seatbelts flagged with flag in model m:

```
Seatbelt2D.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

Seatbelt2D object.

Return type

Seatbelt2D

Example

To check if Seatbelt2D property s.example is a parameter by using the Seatbelt2D.GetParameter() method:

```
if (s.ViewParameters().GetParameter(s.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for seatbelt with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The seatbelt object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...

autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for seatbelt s:

```
s.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for seatbelt s:

```
s.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for seatbelt. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for seatbelt s:

```
s.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this seatbelt.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for seatbelt s:

```
var xrefs = s.Xrefs();
```

---

toString()

Description

Creates a string containing the seatbelt data in keyword format. Note that this contains the keyword header and the keyword cards. See also Seatbelt2D.Keyword() and Seatbelt2D.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for seatbelt s in keyword format

```
var str = s.toString();
```

Sensor class

The Sensor class gives you access to seatbelt sensor cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- ExtractColour()
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)

- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

Sensor properties

Name	Type	Description
acc	real	Activating acceleration.
atime	real	Time over which acceleration must be exceeded.
colour	Colour	The colour of the sensor
dmn	real	Minimum distance
dmx	real	Maximum distance
dof	integer	Degree of freedom.
exists (read only)	logical	true if sensor exists, false if referred to but not defined.
include	integer	The Include file number that the sensor is in.
label	integer	Sensor number. Also see the sbacid property which is an alternative name for this.
model (read only)	integer	The Model number that the sensor is in.
nid	integer	Node number.
nid1	integer	Node number 1
nid2	integer	Node number 2
pulmn	real	Maximum pull-out
pulmx	real	Maximum pull-out
pulrat	real	Rate of pull-out (length/time units)
pultim	real	Time over which rate of pull#out must be exceeded
sbrid	integer	Retractor number (for sbstyp = 2 OR 5).
sbsfl	integer	Sensor flag.
sbsid	integer	Sensor number. Also see the label property which is an alternative name for this.
sbstyp	integer	Sensor type.
time	real	Time at which sensor triggers
transparency	integer	The transparency of the sensor (0-100) 0% is opaque, 100% is transparent.

Detailed Description

The Sensor class allows you to create, modify, edit and manipulate seatbelt sensor cards. See the documentation below for more details.

Constructor

new Sensor(Model*[Model]*, sbsid*[integer]*, sbstyp*[integer]*, sbsfl (optional)*[integer]*, nid (optional)*[integer]*, nid2 (optional)*[integer]*)

Description

Create a new Seatbelt Sensor object.

Arguments

- **Model** (Model)

Model that sensor will be created in

- **sbsid** (integer)

Sensor number.

- **sbstyp** (integer)

Sensor type

- **sbsfl (optional)** (integer)

Sensor flag. Default 0.

- **nid (optional)** (integer)

Optional node ID: Compulsory for types 1 and 4.

- **nid2 (optional)** (integer)

Optional node ID 2: Compulsory for type 4.

Returns

Sensor object

Return type

Sensor

Example

To create a new seatbelt sensor in model m with label 100, type 1 and node 1:

```
var s = new Sensor(m, 100, 1, 0, 1);
```

Details of functions

AssociateComment(Comment[*Comment*])

Description

Associates a comment with a sensor.

Arguments

- **Comment** (Comment)

Comment that will be attached to the sensor

Returns

No return value

Example

To associate comment c to the sensor s:

```
s.AssociateComment(c);
```

Blank()

Description

Blanks the sensor

Arguments

No arguments

Returns

No return value

Example

To blank sensor s:

```
s.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the sensors in the model.

Arguments

- **Model** (Model)

Model that all sensors will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the sensors in model m:

```
Sensor.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged sensors in the model.

Arguments

- **Model** (Model)

Model that all the flagged sensors will be blanked in

- **flag** (Flag)

Flag set on the sensors that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the sensors in model m flagged with f:

```
Sensor.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the sensor is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if sensor s is blanked:

```
if (s.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse sensor s:

```
s.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the sensor.

Arguments

- **flag** (Flag)

Flag to clear on the sensor

Returns

No return value

Example

To clear flag f for sensor s:

```
s.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the sensor. The target include of the copied sensor can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

Sensor object

Return type

Sensor

Example

To copy sensor s into sensor z:

```
var z = s.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a sensor

Arguments

- **Model** (Model)

Model that the sensor will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

Sensor object (or null if not made).

Return type

Sensor

Example

To start creating a sensor in model m:

```
var s = Sensor.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a sensor.

Arguments

- **Comment** (Comment)

Comment that will be detached from the sensor

Returns

No return value

Example

To detach comment c from the sensor s:

```
s.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit sensor s:

```
s.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for sensor with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description

autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The sensor object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for sensor s:

```
s.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for sensor s:

```
s.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for sensor. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for sensor s:

```
s.Error("My custom error");
```

ExtractColour()

Description

Extracts the **actual** colour used for sensor.
By default in PRIMER many entities such as elements get their colour automatically from the part that they are in. PRIMER cycles through 13 default colours based on the label of the entity. In this case the sensor colour property will return the value Colour.PART instead of the actual colour. This method will return the actual colour which is used for drawing the sensor.

Arguments

No arguments

Returns

colour value (integer)

Return type

Number

Example

To return the colour used for drawing sensor s:

```
var colour = s.ExtractColour();
```

First(Model*[Model]*) [static]

Description

Returns the first sensor in the model.

Arguments

- **Model** (Model)

Model to get first sensor in

Returns

Sensor object (or null if there are no sensors in the model).

Return type

Sensor

Example

To get the first sensor in model m:

```
var s = Sensor.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free sensor label in the model. Also see Sensor.LastFreeLabel(), Sensor.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free sensor label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

Sensor label.

Return type

Number

Example

To get the first free sensor label in model m:

```
var label = Sensor.FirstFreeLabel(m);
```

FlagAll(Model[*Model*], flag[*Flag*]) [static]

Description

Flags all of the sensors in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all sensors will be flagged in

- **flag** (Flag)

Flag to set on the sensors

Returns

No return value

Example

To flag all of the sensors with flag f in model m:

```
Sensor.FlagAll(m, f);
```

Flagged(flag[*Flag*])

Description

Checks if the sensor is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the sensor

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if sensor s has flag f set on it:

```
if (s.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each sensor in the model.
Note that ForEach has been designed to make looping over sensors as fast as possible and so has some limitations.
Firstly, a single temporary Sensor object is created and on each function call it is updated with the current sensor data. This means that you should not try to store the Sensor object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new sensors inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all sensors are in

- **func** (function)

Function to call for each sensor

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the sensors in model m:

```
Sensor.ForEach(m, test);
function test(s)
{
// s is Sensor object
}
```

To call function test for all of the sensors in model m with optional object:

```
var data = { x:0, y:0 };
Sensor.ForEach(m, test, data);
function test(s, extra)
{
// s is Sensor object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Sensor objects or properties for all of the sensors in a model in PRIMER. If the optional property argument is not given then an array of Sensor objects is returned. If the property argument is given, that property value for each sensor is returned in the array instead of a Sensor object

Arguments

- **Model** (Model)

Model to get sensors from

- **property (optional)** (string)

Name for property to get for all sensors in the model

Returns

Array of Sensor objects or properties

Return type

Array

Example

To make an array of Sensor objects for all of the sensors in model m:

```
var a = Sensor.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each sensor in model m:

```
var a = Sensor.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a sensor.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the sensor s:

```
var comm_array = s.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of Sensor objects for all of the flagged sensors in a model in PRIMER If the optional property argument is not given then an array of Sensor objects is returned. If the property argument is given, then that property value for each sensor is returned in the array instead of a Sensor object

Arguments

- **Model** (Model)

Model to get sensors from

- **flag** (Flag)

Flag set on the sensors that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged sensors in the model

Returns

Array of Sensor objects or properties

Return type

Array

Example

To make an array of Sensor objects for all of the sensors in model m flagged with f:

```
var s = Sensor.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the sensors in model m flagged with f:

```
var a = Sensor.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the Sensor object for a sensor ID.

Arguments

- **Model** (Model)

Model to find the sensor in

- **number** (integer)

number of the sensor you want the Sensor object for

Returns

Sensor object (or null if sensor does not exist).

Return type

Sensor

Example

To get the Sensor object for sensor 100 in model m

```
var s = Sensor.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a Sensor property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the Sensor.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

sensor property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if Sensor property s.example is a parameter:

```
Options.property_parameter_names = true;
if (s.GetParameter(s.example) ) do_something...
Options.property_parameter_names = false;
```

To check if Sensor property s.example is a parameter by using the GetParameter method:

```
if (s.ViewParameters().GetParameter(s.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this sensor (*ELEMENT_SEATBELT_SENSEROMETER) **Note that a carriage return is not added**. See also Sensor.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for sensor s:

```
var key = s.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the sensor. **Note that a carriage return is not added**. See also Sensor.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for sensor s:

```
var cards = s.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last sensor in the model.

Arguments

- **Model** (Model)

Model to get last sensor in

Returns

Sensor object (or null if there are no sensors in the model).

Return type

Sensor

Example

To get the last sensor in model m:

```
var s = Sensor.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free sensor label in the model. Also see Sensor.FirstFreeLabel(), Sensor.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free sensor label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

Sensor label.

Return type

Number

Example

To get the last free sensor label in model m:

```
var label = Sensor.LastFreeLabel(m);
```

Next()

Description

Returns the next sensor in the model.

Arguments

No arguments

Returns

Sensor object (or null if there are no more sensors in the model).

Return type

Sensor

Example

To get the sensor in model m after sensor s:

```
var s = s.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) sensor label in the model. Also see Sensor.FirstFreeLabel(), Sensor.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free sensor label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

Sensor label.

Return type

Number

Example

To get the next free sensor label in model m:

```
var label = Sensor.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a sensor.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only sensors from that model can be picked. If the argument is a Flag then only sensors that are flagged with *limit* can be selected. If omitted, or null, any sensors from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

Sensor object (or null if not picked)

Return type

Sensor

Example

To pick a sensor from model m giving the prompt 'Pick sensor from screen':

```
var s = Sensor.Pick('Pick sensor from screen', m);
```

Previous()

Description

Returns the previous sensor in the model.

Arguments

No arguments

Returns

Sensor object (or null if there are no more sensors in the model).

Return type

Sensor

Example

To get the sensor in model m before sensor s:

```
var s = s.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the sensors in the model.

Arguments

- **Model** (Model)

Model that all sensors will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the sensors in model m, from 1000000:

```
Sensor.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged sensors in the model.

Arguments

- **Model** (Model)

Model that all the flagged sensors will be renumbered in

- **flag** (Flag)

Flag set on the sensors that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the sensors in model m flagged with f, from 1000000:

```
Sensor.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select sensors using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting sensors

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only sensors from that model can be selected. If the argument is a Flag then only sensors that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any sensors can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of sensors selected or null if menu cancelled

Return type

Number

Example

To select sensors from model m, flagging those selected with flag f, giving the prompt 'Select sensors':

```
Sensor.Select(f, 'Select sensors', m);
```

To select sensors, flagging those selected with flag f but limiting selection to sensors flagged with flag l, giving the prompt 'Select sensors':

```
Sensor.Select(f, 'Select sensors', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the sensor.

Arguments

- **flag** (Flag)

Flag to set on the sensor

Returns

No return value

Example

To set flag f for sensor s:

```
s.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the sensor. The sensor will be sketched until you either call Sensor.Unsketch(), Sensor.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the sensor is sketched. If omitted redraw is true. If you want to sketch several sensors and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch sensor s:

```
s.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged sensors in the model. The sensors will be sketched until you either call Sensor.Unsketch(), Sensor.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged sensors will be sketched in

- **flag** (Flag)

Flag set on the sensors that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the sensors are sketched. If omitted redraw is true. If you want to sketch flagged sensors several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all sensors flagged with flag in model m:

```
Sensor.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of sensors in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing sensors should be counted. If false or omitted referenced but undefined sensors will also be included in the total.

Returns

number of sensors

Return type

Number

Example

To get the total number of sensors in model m:

```
var total = Sensor.Total(m);
```

Unblank()

Description

Unblanks the sensor

Arguments

No arguments

Returns

No return value

Example

To unblank sensor s:

```
s.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the sensors in the model.

Arguments

- **Model** (Model)

Model that all sensors will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the sensors in model m:

```
Sensor.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged sensors in the model.

Arguments

- **Model** (Model)

Model that the flagged sensors will be unblanked in

- **flag** (Flag)

Flag set on the sensors that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the sensors in model m flagged with f:

```
Sensor.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the sensors in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all sensors will be unset in

- **flag** (Flag)

Flag to unset on the sensors

Returns

No return value

Example

To unset the flag f on all the sensors in model m:

```
Sensor.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the sensor.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the sensor is unsketched. If omitted redraw is true. If you want to unsketch several sensors and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch sensor s:

```
s.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all sensors.

Arguments

- **Model** (Model)

Model that all sensors will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the sensors are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all sensors in model m:

```
Sensor.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged sensors in the model.

Arguments

- **Model** (Model)

Model that all sensors will be unsketched in

- **flag** (Flag)

Flag set on the sensors that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the sensors are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all sensors flagged with flag in model m:

```
Sensor.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

Sensor object.

Return type

Sensor

Example

To check if Sensor property s.example is a parameter by using the Sensor.GetParameter() method:

```
if (s.ViewParameters().GetParameter(s.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for sensor with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The sensor object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for sensor s:

```
s.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for sensor s:

```
s.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for sensor. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for sensor s:

```
s.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this sensor.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for sensor s:

```
var xrefs = s.Xrefs();
```

---

toString()

Description

Creates a string containing the sensor data in keyword format. Note that this contains the keyword header and the keyword cards. See also Sensor.Keyword() and Sensor.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for sensor s in keyword format

```
var str = s.toString();
```

Shell class

The Shell class gives you access to shell cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- FillHolesOnFlagged(Model*[Model]*, Flag*[Flag]*, RemeshHole*[boolean]*, pid (optional)*[integer]*, Max Hole Size (optional)*[real]*, Mesh Element size (optional)*[real]*, planarSurface (optional)*[boolean]*)
- FindShellInBox(Model*[Model]*, xmin*[real]*, xmax*[real]*, ymin*[real]*, ymax*[real]*, zmin*[real]*, zmax*[real]*, flag (optional)*[integer]*, excl (optional)*[integer]*, vis_only (optional)*[integer]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- MakeConsistentNormalsFlagged(Model*[Model]*, Flag*[Flag]*, Shell label (optional)*[integer]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- PickIsoparametric(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- ReverseNormalsFlagged(Model*[Model]*, Flag*[Flag]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- Angles()
- Area()
- AspectRatio()
- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- CoordsToIsoparametric(x*[real]*, y*[real]*, z*[real]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- ElemCut(Database cross section label*[integer]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- ExtractColour()

- FillAttachedHole(pid*[integer]*, size*[real]*)
- Flagged(flag*[Flag]*)
- GetAttachedShells(tolerance (optional)*[real]*, recursive (optional)*[boolean]*)
- GetComments()
- GetCompositeData(ipt*[integer]*)
- GetNodeIDs()
- GetNodes()
- GetParameter(prop*[string]*)
- GetShellReferenceGeometry()
- IsoparametricToCoords(s*[real]*, t*[real]*)
- Jacobian()
- Keyword()
- KeywordCards()
- Length()
- Next()
- NormalVector()
- Previous()
- RemoveCompositeData(ipt*[integer]*)
- ReverseNormal(redraw (optional)*[boolean]*)
- SetCompositeData(ipt*[integer]*, mid*[integer]*, thick*[real]*, beta*[real]*, plyid (optional)*[integer]*)
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Skew()
- Taper()
- TiedNodeCheck(Contact label*[integer]*, Flag*[Flag]*, Option1*[integer]*, Option2*[integer]*)
- Timestep()
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Warpage()
- WeightingFactors(s*[real]*, t*[real]*)
- Xrefs()
- toString()

Shell constants

Name	Description
Shell.EDGE_1	Edge 1 of shell
Shell.EDGE_2	Edge 2 of shell
Shell.EDGE_3	Edge 3 of shell
Shell.EDGE_4	Edge 4 of shell

Shell properties

Name	Type	Description
beta	real	Orthotropic material base offset angle. null if the _BETA option is not set. If not null then this is the angle in degrees and the _BETA option is set. This is required to distinguish between the cases of _BETA not being used (beta === null) and _BETA being set but the angle being zero (beta === 0). Prior to version 18 _BETA was only set if beta was non-zero. This was fixed in version 18 and the test changed to beta not being null. **Note: If this option is set then mcid should be 0**
colour	Colour	The colour of the shell
composite	logical	If COMPOSITE option is set. Can be true or false
composite_long	logical	If COMPOSITE_LONG option is set. Can be true or false
dof	logical	If DOF option is set. Can be true or false

edges	constant	Bitwise code of Shell.EDGE_1, Shell.EDGE_2, Shell.EDGE_3 and Shell.EDGE_4 representing which edges of the shell are free edges
eid	integer	Shell number. Also see the label property which is an alternative name for this.
exists (read only)	logical	true if shell exists, false if referred to but not defined.
include	integer	The Include file number that the shell is in.
label	integer	Shell number. Also see the eid property which is an alternative name for this.
mcid	integer	Material coordinate system ID. If non zero then the _MCID option is assumed. **Note: If this option is set then beta should be null.**
model (read only)	integer	The Model number that the shell is in.
n1	integer	Node number 1
n2	integer	Node number 2
n3	integer	Node number 3
n4	integer	Node number 4
n5	integer	Node number 5
n6	integer	Node number 6
n7	integer	Node number 7
n8	integer	Node number 8
nip	logical	Number of integration points for composite shell
nodes (read only)	integer	Number of nodes shell has
ns1	integer	Scalar Node number 1
ns2	integer	Scalar Node number 2
ns3	integer	Scalar Node number 3
ns4	integer	Scalar Node number 4
offset	real	Offset distance. If non zero then the _OFFSET option is assumed
pid	integer	Part number
shl4_to_shl8	logical	If SHL4_TO_SHL8 option is set. Can be true or false
thic1	real	Thickness at node 1
thic2	real	Thickness at node 2
thic3	real	Thickness at node 3
thic4	real	Thickness at node 4
thic5	real	Thickness at node 5 (if 8 noded shell)
thic6	real	Thickness at node 6 (if 8 noded shell)
thic7	real	Thickness at node 7 (if 8 noded shell)
thic8	real	Thickness at node 8 (if 8 noded shell)
thickness	logical	If _THICKNESS option is set. Can be true or false
transparency	integer	The transparency of the shell (0-100) 0% is opaque, 100% is transparent.

Detailed Description

The Shell class allows you to create, modify, edit and manipulate shell cards. See the documentation below for more details.

Constructor

new Shell(Model*[Model]*, eid*[integer]*, pid*[integer]*, n1*[integer]*, n2*[integer]*, n3*[integer]*, n4 (optional)*[integer]*, n5 (optional)*[integer]*, n6 (optional)*[integer]*, n7 (optional)*[integer]*, n8 (optional)*[integer]*)

Description

Create a new Shell object. Use either 3, 4, 6 or 8 nodes when creating a new shell. If you are creating a 3 noded shell either only give 3 nodes or give 4 nodes but make nodes 3 and 4 the same number. Similarly, 6 noded shells can be created with 6 node arguments or with 8 nodes but nodes 3 and 4 the same number and nodes 7 and 8 the same number.

Arguments

- **Model** (Model)

Model that shell will be created in

- **eid** (integer)

Shell number

- **pid** (integer)

Part number

- **n1** (integer)

Node number 1

- **n2** (integer)

Node number 2

- **n3** (integer)

Node number 3

- **n4 (optional)** (integer)

Node number 4

- **n5 (optional)** (integer)

Node number 5

- **n6 (optional)** (integer)

Node number 6

- **n7 (optional)** (integer)

Node number 7

- **n8 (optional)** (integer)

Node number 8

Returns

Shell object

Return type

Shell

Example

To create a new shell in model m with label 100, part 10 and nodes 1, 2, 3, 4:

```
var s = new Shell(m, 100, 10, 1, 2, 3, 4);
```

Details of functions

Angles()

Description

Calculates the minimum and maximum internal angles (in degrees) for the shell

Arguments

No arguments

Returns

Array of numbers containing min and max angles

Return type

Number

Example

To calculate the maximum and minimum internal angles for shell s:

```
var angles = s.Angles();
var min = angles[0];
var max = angles[1];
```

Area()

Description

Calculates the area for the shell

Arguments

No arguments

Returns

real

Return type

Number

Example

To calculate the area for shell s:

```
var area = s.Area();
```

AspectRatio()

Description

Calculates the aspect ratio for the shell

Arguments

No arguments

Returns

real

Return type

Number

Example

To calculate the aspect ratio for shell s:

```
var ratio = s.AspectRatio();
```

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a shell.

Arguments

- **Comment** (Comment)

Comment that will be attached to the shell

Returns

No return value

Example

To associate comment c to the shell s:

```
s.AssociateComment(c);
```

Blank()

Description

Blanks the shell

Arguments

No arguments

Returns

No return value

Example

To blank shell s:

```
s.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the shells in the model.

Arguments

- **Model** (Model)

Model that all shells will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the shells in model m:

```
Shell.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged shells in the model.

Arguments

- **Model** (Model)

Model that all the flagged shells will be blanked in

- **flag** (Flag)

Flag set on the shells that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the shells in model m flagged with f:

```
Shell.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the shell is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if shell s is blanked:

```
if (s.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse shell s:

```
s.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the shell.

Arguments

- **flag** (Flag)

Flag to clear on the shell

Returns

No return value

Example

To clear flag f for shell s:

```
s.ClearFlag(f);
```

CoordsToIsoparametric(x*[real]*, y*[real]*, z*[real]*)

Description

Calculates the isoparametric coordinates for a point on the shell.

Arguments

- **x** (real)

X coordinate of point

- **y** (real)

Y coordinate of point

- **z** (real)

Z coordinate of point

Returns

Array containing s and t isoparametric coordinates and the distance the point is from the shell (positive in direction of shell normal). If it is not possible to calculate the isoparametric coordinates null is returned.

Return type

Array

Example

To calculate the isoparametric coordinates of point (10, 20, 30) on shell s:

```
var isocoords = s.CoordsToIsoparametric(10, 20, 30);
```

Copy(range (optional)*[boolean]*)

Description

Copies the shell. The target include of the copied shell can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

Shell object

Return type

Shell

Example

To copy shell s into shell z:

```
var z = s.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a shell

Arguments

- **Model** (Model)

Model that the shell will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

Shell object (or null if not made).

Return type

Shell

Example

To start creating a shell in model m:

```
var s = Shell.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a shell.

Arguments

- **Comment** (Comment)

Comment that will be detached from the shell

Returns

No return value

Example

To detach comment c from the shell s:

```
s.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit shell s:

```
s.Edit();
```

ElemCut(Database cross section label*[integer]*)

Description

Returns coordinates of the intersections between a shell and a database cross section.

Arguments

- **Database cross section label** (integer)

The label of the database cross section.

Returns

An array containing the x1,y1,z1,x2,y2,z2 coordinates of the cut line, or NULL if it does not cut. Note this function does not check that the shell is in the cross section definition (part set)

Return type

Array

Example

To get the cut line coordinates between database cross section 200 and shell s:

```
var data = s.ElemCut(200)
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for shell with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The shell object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for shell s:

```
s.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for shell s:

```
s.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for shell. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for shell s:

```
s.Error("My custom error");
```

ExtractColour()

Description

Extracts the **actual** colour used for shell.
By default in PRIMER many entities such as elements get their colour automatically from the part that they are in. PRIMER cycles through 13 default colours based on the label of the entity. In this case the shell colour property will return the value Colour.PART instead of the actual colour. This method will return the actual colour which is used for drawing the shell.

Arguments

No arguments

Returns

colour value (integer)

Return type

Number

Example

To return the colour used for drawing shell s:

```
var colour = s.ExtractColour();
```

FillAttachedHole(pid*[integer]*, size*[real]*)

Description

Fills in (meshes) a hole attached to the shell.

Arguments

- **pid** (integer)

The Part number that the new shells will be created in.

- **size** (real)

The size for created elements.

Returns

No return value.

Example

To fill in a hole attached to shell s, putting new shells with size 5.0 into part 100:

```
s.FillAttachedHole(100, 5.0);
```

FillHolesOnFlagged(Model*[Model]*, Flag*[Flag]*, RemeshHole*[boolean]*, pid (optional)*[integer]*, Max Hole Size (optional)*[real]*, Mesh Element size (optional)*[real]*, planarSurface (optional)*[boolean]*) [static]

Description

Fills multiple holes using flagged shells.

Arguments

- **Model** (Model)

Model that all shells are in.

- **Flag** (Flag)

flag bit

- **RemeshHole** (boolean)

TRUE if elements around the hole should be remeshed

- **pid (optional)** (integer)

Needs to be specified if RemeshHole is FALSE. Specifies the Part id where the mesh is filled

- **Max Hole Size (optional)** (real)

Maximum size of the hole which is to be filled. If omitted a default size of 20.0 will be set

- **Mesh Element size (optional)** (real)

Element size of the mesh which fills the hole. If omitted a default size of 10.0 will be set

- **planarSurface (optional)** (boolean)

Needs to be specified if RemeshHole is TRUE. TRUE if we need to Use planar surface

Returns

No return value.

Example

To fill holes on flagged shells:

```
Shell.FillHolesOnFlagged(m, flag, 1, 112, 60.5, 5.34 ,0);
```

Note: pid is required when RemeshHole is FALSE

```
Shell.FillHolesOnFlagged(m, flag, 0, 112);
```

FindShellInBox(Model*[Model]*, xmin*[real]*, xmax*[real]*, ymin*[real]*, ymax*[real]*, zmin*[real]*, zmax*[real]*, flag (optional)*[integer]*, excl (optional)*[integer]*, vis_only (optional)*[integer]*) [static]

Description

Returns an array of Shell objects for the shells within a box. Please note in (default) inclusive mode this function provides a list of all shells that could potentially be in the box (using computationally cheap bounding box comparison - local box vs main box). NOTE - it is not a rigorous test of whether the shell is actually in the box. An extension of "spot_thickness" is applied to each local shell box. By default this is 10mm. You can use "Options.connection_max_ thickness = x" to reduce this value. This may return shells that are ostensibly outside box. The user should apply their own test on each shell returned. The purpose of this function is to reduce the number of shells you need to test. Setting the exclusive option will only return shells that are fully contained in the main box This may not capture all the shells you want to process so must be used with care.

Arguments

- **Model** (Model)

Model designated model

- **xmin** (real)

Minimum bound in global x

- **xmax** (real)

Maximum bound in global x

- **ymin** (real)

Minimum bound in global y

- **ymax** (real)

Maximum bound in global y

- **zmin** (real)

Minimum bound in global z

- **zmax** (real)

Maximum bound in global z

- **flag (optional)** (integer)

Optional flag to restrict shells considered, if 0 all shells considered

- **excl (optional)** (integer)

Optional flag (0) Apply inclusive selection with local box extension = "spot_thickness" (default 10) (-1) Apply inclusive selection with local box extension = 0.5*shell thickness (1) Apply exclusive selection inclusive selection means elements intersect box exclusive selection means elements contained in box

- **vis_only (optional)** (integer)

Optional flag to consider visible shells only (1), if (0) all shells considered

Returns

Array of Shell objects

Return type

Array

Example

To get an array of Shell objects for flagged shells within defined box.

```
var s = Shell.FindShellInBox(m, xmin, xmax, ymin, ymax, zmin, zmax, flag, 0, 0);
```

First(Model*[Model]*) [static]

Description

Returns the first shell in the model.

Arguments

- **Model** (Model)

Model to get first shell in

Returns

Shell object (or null if there are no shells in the model).

Return type

Shell

Example

To get the first shell in model m:

```
var s = Shell.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free shell label in the model. Also see Shell.LastFreeLabel(), Shell.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free shell label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

Shell label.

Return type

Number

Example

To get the first free shell label in model m:

```
var label = Shell.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the shells in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all shells will be flagged in

- **flag** (Flag)

Flag to set on the shells

Returns

No return value

Example

To flag all of the shells with flag f in model m:

```
Shell.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the shell is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the shell

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if shell s has flag f set on it:

```
if (s.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each shell in the model.
Note that ForEach has been designed to make looping over shells as fast as possible and so has some limitations. Firstly, a single temporary Shell object is created and on each function call it is updated with the current shell data. This means that you should not try to store the Shell object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new shells inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all shells are in

- **func** (function)

Function to call for each shell

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the shells in model m:

```
Shell.ForEach(m, test);
function test(s)
{
// s is Shell object
}
```

To call function test for all of the shells in model m with optional object:

```
var data = { x:0, y:0 };
Shell.ForEach(m, test, data);
function test(s, extra)
{
// s is Shell object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Shell objects or properties for all of the shells in a model in PRIMER. If the optional property argument is not given then an array of Shell objects is returned. If the property argument is given, that property value for each shell is returned in the array instead of a Shell object

Arguments

- **Model** (Model)

Model to get shells from

- **property (optional)** (string)

Name for property to get for all shells in the model

Returns

Array of Shell objects or properties

Return type

Array

Example

To make an array of Shell objects for all of the shells in model m:

```
var a = Shell.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each shell in model m:

```
var a = Shell.GetAll(m, 'foo');
```

GetAttachedShells(tolerance (optional)*[real]*, recursive (optional)*[boolean]*)

Description

Returns the shells that are attached to the shell. **Note that 'attached' means that the shells must share 2 nodes**.

Arguments

- **tolerance (optional)** (real)

This tolerance can be used to limit the selection to shells whose normal vector is within this tolerance (in degrees) of the original shell. If omitted the tolerance is 180 degrees.

- **recursive (optional)** (boolean)

If recursive is false then only the shells actually attached to the shell will be returned (this could also be done by using

the Xrefs class but this method is provided for convenience. If recursive is true then PRIMER will keep finding attached shells until no more can be found. If omitted recursive will be false.

Returns

Array of Shell objects (or null if there are no attached shells).

Return type

Array

Example

To find the shells attached to shell s with a 10 degree tolerance, growing the selection until no more shells can be found:

```
var shell_array = s.GetAttachedShells(10, true);
```

GetComments()

Description

Extracts the comments associated to a shell.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the shell s:

```
var comm_array = s.GetComments();
```

GetCompositeData(ipt*[integer]*)

Description

Returns the composite data for an integration point in *ELEMENT_SHELL_COMPOSITE.

Arguments

- **ipt** (integer)

The integration point you want the data for. **Note that integration points start at 0, not 1**.

Returns

An array containing the material ID, thickness and beta angle values. If the _COMPOSITE_LONG option is set, then the array returned will also contain the ply ID.

Return type

Array

Example

To get the composite data for the 3rd integration point for shell s:

```
if (s.composite && s.nip >= 3)
{
    var ipt_data = s.GetCompositeData(2);
}
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of Shell objects for all of the flagged shells in a model in PRIMER If the optional property argument is not given then an array of Shell objects is returned. If the property argument is given, then that property value for each shell is returned in the array instead of a Shell object

Arguments

- **Model** (Model)

Model to get shells from

- **flag** (Flag)

Flag set on the shells that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged shells in the model

Returns

Array of Shell objects or properties

Return type

Array

Example

To make an array of Shell objects for all of the shells in model m flagged with f:

```
var s = Shell.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the shells in model m flagged with f:

```
var a = Shell.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the Shell object for a shell ID.

Arguments

- **Model** (Model)

Model to find the shell in

- **number** (integer)

number of the shell you want the Shell object for

Returns

Shell object (or null if shell does not exist).

Return type

Shell

Example

To get the Shell object for shell 100 in model m

```
var s = Shell.GetFromID(m, 100);
```

GetNodeIDs()

Description

Returns the labels of the nodes on the shell as an array. See also Shell.GetNodes()

Arguments

No arguments

Returns

Array of node labels (integers)

Return type

Number

Example

To return the node labels of shell s as an array

```
var nodes = s.GetNodeIDs();
```

GetNodes()

Description

Returns the nodes on the shell as an array of Node objects. See also Shell.GetNodeIDs()

Arguments

No arguments

Returns

Array of Node objects

Return type

Array

Example

To return the nodes of shell s as an array

```
var nodes = s.GetNodes();
```

GetParameter(prop*[string]*)

Description

Checks if a Shell property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the Shell.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

shell property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if Shell property s.example is a parameter:

```
Options.property_parameter_names = true;
if (s.GetParameter(s.example) ) do_something...
Options.property_parameter_names = false;
```

To check if Shell property s.example is a parameter by using the GetParameter method:

```
if (s.ViewParameters().GetParameter(s.example) ) do_something...
```

GetShellReferenceGeometry()

Description

Returns the airbag shell reference geometry of the shell

Arguments

No arguments

Returns

The shell reference geometry ID of the shell (or 0 if it hasn't got any)

Return type

Number

Example

To get the shell reference geometry of the shell s:

```
var a = s.GetShellReferenceGeometry();
```

IsoparametricToCoords(s*[real]*, t*[real]*)

Description

Calculates the coordinates for a point on the shell from the isoparametric coords.

Arguments

- **s** (real)

First isoparametric coordinate

- **t** (real)

Second isoparametric coordinate

Returns

Array of numbers containing x, y and z or null if not possible to calculate.

Return type

Number

Example

To calculate the coordinates of isoparametric point (0.5, -0.5) on shell s:

```
var coords = s.IsoparametricToCoords(0.5, -0.5);
```

Jacobian()

Description

Calculates the jacobian for the shell

Arguments

No arguments

Returns

real

Return type

Number

Example

To calculate the jacobian for shell s:

```
var jacobian = s.Jacobian();
```

Keyword()

Description

Returns the keyword for this shell (*SHELL, *SHELL_SCALAR or *SHELL_SCALAR_VALUE). **Note that a carriage return is not added**. See also Shell.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for shell s:

```
var key = s.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the shell. **Note that a carriage return is not added**. See also Shell.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for shell s:

```
var cards = s.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last shell in the model.

Arguments

- **Model** (Model)

Model to get last shell in

Returns

Shell object (or null if there are no shells in the model).

Return type

Shell

Example

To get the last shell in model m:

```
var s = Shell.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free shell label in the model. Also see Shell.FirstFreeLabel(), Shell.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free shell label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

Shell label.

Return type

Number

Example

To get the last free shell label in model m:

```
var label = Shell.LastFreeLabel(m);
```

Length()

Description

Calculates the minimum length for the shell

Arguments

No arguments

Returns

real

Return type

Number

Example

To calculate the minimum length for shell s:

```
var length = s.Length();
```

MakeConsistentNormalsFlagged(Model*[Model]*, Flag*[Flag]*, Shell label (optional)*[integer]*) [static]

Description

Make all the flagged SHELL normals consistent with a selected one, the Seed Element.

Arguments

- **Model** (Model)

Model that all shells are in.

- **Flag** (Flag)

flag bit

- **Shell label (optional)** (integer)

The label of the seed shell. If omitted, or null, the first flagged shell is used as the seed shell.

Returns

Array containing the labels of shells which have had normals reversed

Return type

Array

Example

To make all flagged shell normals consistent:

```
Shell.MakeConsistentNormalsFlagged(m, flag, 1001);
```

Next()

Description

Returns the next shell in the model.

Arguments

No arguments

Returns

Shell object (or null if there are no more shells in the model).

Return type

Shell

Example

To get the shell in model m after shell s:

```
var s = s.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) shell label in the model. Also see Shell.FirstFreeLabel(), Shell.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free shell label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

Shell label.

Return type

Number

Example

To get the next free shell label in model m:

```
var label = Shell.NextFreeLabel(m);
```

NormalVector()

Description

Calculates the unit normal vector for the shell.

Arguments

No arguments

Returns

Array of numbers containing x, y and z components of unit normal vector or null if the vector cannot be calculated (for example if the shell has zero area).

Return type

Number

Example

To calculate the normal vector of shell s:

```
var nvector = s.NormalVector();
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a shell.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only shells from that model can be picked. If the argument is a Flag then only shells that are flagged with *limit* can be selected. If omitted, or null, any shells from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

Shell object (or null if not picked)

Return type

Shell

Example

To pick a shell from model m giving the prompt 'Pick shell from screen':

```
var s = Shell.Pick('Pick shell from screen', m);
```

PickIsoparametric(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a point on a shell. The isoparametric coordinates of the point picked on the shell are returned as well as the shell picked. These coordinates are suitable for using in the function Shell.IsoparametricToCoords(). See also Shell.Pick()

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only shells from that model can be picked. If the argument is a Flag then only shells that are flagged with *limit* can be selected. If omitted, or null, any shells from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

Array containing Shell object and isoparametric coordinates (or null if not picked or the point is not on a shell)

Return type

Array

Example

To pick a point on a shell from model m giving the prompt 'Pick a point on a shell on the screen':

```
var a = Shell.PickIsoparametric('Pick a point on a shell on the screen', m);
if (a != null)
{
    Message("You picked point "+a[1]+","+a[2]+" on shell "+a[0].label);
}
```

Previous()

Description

Returns the previous shell in the model.

Arguments

No arguments

Returns

Shell object (or null if there are no more shells in the model).

Return type

Shell

Example

To get the shell in model m before shell s:

```
var s = s.Previous();
```

RemoveCompositeData(ipt*[integer]*)

Description

Removes the composite data for an integration point in *ELEMENT_SHELL_COMPOSITE.

Arguments

- **ipt** (integer)

The integration point you want to remove. **Note that integration points start at 0, not 1**.

Returns

No return value.

Example

To remove the composite data for the 3rd integration point for shell s:

```
s.RemoveCompositeData(2);
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the shells in the model.

Arguments

- **Model** (Model)

Model that all shells will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the shells in model m, from 1000000:

```
Shell.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged shells in the model.

Arguments

- **Model** (Model)

Model that all the flagged shells will be renumbered in

- **flag** (Flag)

Flag set on the shells that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the shells in model m flagged with f, from 1000000:

```
Shell.RenumberFlagged(m, f, 1000000);
```

ReverseNormal(redraw (optional)*[boolean]*)

Description

Reverse shell normal.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to reverse several shell normals and only redraw after the last one then use false for all redraws apart from the last one.

Returns

No return value.

Example

To Reverse shell normal for shell s:

```
s.ReverseNormal();
```

ReverseNormalsFlagged(Model*[Model]*, Flag*[Flag]*) [static]

Description

Reverse all the flagged shell normals

Arguments

- **Model** (Model)

Model that all shells are in.

- **Flag** (Flag)

flag bit

Returns

No return value.

Example

To Reverse all flagged shell normals:

```
Shell.ReverseNormalsFlagged(m, flag);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select shells using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting shells

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only shells from that model can be selected. If the argument is a Flag then only shells that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any shells can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of shells selected or null if menu cancelled

Return type

Number

Example

To select shells from model m, flagging those selected with flag f, giving the prompt 'Select shells':

```
Shell.Select(f, 'Select shells', m);
```

To select shells, flagging those selected with flag f but limiting selection to shells flagged with flag l, giving the prompt 'Select shells':

```
Shell.Select(f, 'Select shells', l);
```

SetCompositeData(ipt*[integer]*, mid*[integer]*, thick*[real]*, beta*[real]*, plyid (optional)*[integer]*)

Description

Sets the composite data for an integration point in *ELEMENT_SHELL_COMPOSITE.

Arguments

- **ipt** (integer)

The integration point you want to set the data for. **Note that integration points start at 0, not 1**.

- **mid** (integer)

Material ID for the integration point.

- **thick** (real)

Thickness of the integration point.

- **beta** (real)

Material angle of the integration point.

- **plyid (optional)** (integer)

Ply ID for the integration point. This should be used if the _COMPOSITE_LONG option is set for the shell.

Returns

No return value.

Example

To set the composite data for the 3rd integration point to mat 1, thickness 0.5 and angle 45, for shell s:

```
s.SetCompositeData(2, 1, 0.5, 45);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the shell.

Arguments

- **flag** (Flag)

Flag to set on the shell

Returns

No return value

Example

To set flag f for shell s:

```
s.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the shell. The shell will be sketched until you either call Shell.Unsketch(), Shell.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the shell is sketched. If omitted redraw is true. If you want to sketch several shells and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch shell s:

```
s.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged shells in the model. The shells will be sketched until you either call Shell.Unsketch(), Shell.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged shells will be sketched in

- **flag** (Flag)

Flag set on the shells that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the shells are sketched. If omitted redraw is true. If you want to sketch flagged shells several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all shells flagged with flag in model m:

```
Shell.SketchFlagged(m, flag);
```

Skew()

Description

Calculates the skew for the shell

Arguments

No arguments

Returns

real

Return type

Number

Example

To calculate the skew for shell s:

```
var skew = s.Skew();
```

Taper()

Description

Calculates the taper for the shell

Arguments

No arguments

Returns

real

Return type

Number

Example

To calculate the taper for shell s:

```
var taper = s.Taper();
```

TiedNodeCheck(Contact label*[integer]*, Flag*[Flag]*, Option1*[integer]*, Option2*[integer]*)

Description

Checks if nodes of shell are tied by contact or directly attached (non-zero option1).

Arguments

- **Contact label** (integer)

The label of the tied contact. If zero the tied contact is found for the shell by reverse lookup.

- **Flag** (Flag)

flag bit

- **Option1** (integer)

Directly tied node (logical OR) 0:NONE 1:NRB/C_EXNO 2:BEAM 4:SHELL 8:SOLID 16:TSHELL

- **Option2** (integer)

0:No action 1: report error if directly attached node (acc. option1) captured by contact

Returns

string

Return type

String

Example

To check if all nodes of shell s are tied by contact 200 or attach directly to constraint:

```
var message = s.TiedNodeCheck(200, flag, 1, 1)
```

Timestep()

Description

Calculates the timestep for the shell

Arguments

No arguments

Returns

real

Return type

Number

Example

To calculate the timestep for shell s:

```
var timestep = s.Timestep();
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of shells in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing shells should be counted. If false or omitted referenced but undefined shells will also be included in the total.

Returns

number of shells

Return type

Number

Example

To get the total number of shells in model m:

```
var total = Shell.Total(m);
```

Unblank()

Description

Unblanks the shell

Arguments

No arguments

Returns

No return value

Example

To unblank shell s:

```
s.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the shells in the model.

Arguments

- **Model** (Model)

Model that all shells will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the shells in model m:

```
Shell.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged shells in the model.

Arguments

- **Model** (Model)

Model that the flagged shells will be unblanked in

- **flag** (Flag)

Flag set on the shells that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the shells in model m flagged with f:

```
Shell.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the shells in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all shells will be unset in

- **flag** (Flag)

Flag to unset on the shells

Returns

No return value

Example

To unset the flag f on all the shells in model m:

```
Shell.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the shell.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the shell is unsketched. If omitted redraw is true. If you want to unsketch several shells and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch shell s:

```
s.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all shells.

Arguments

- **Model** (Model)

Model that all shells will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the shells are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all shells in model m:

```
Shell.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged shells in the model.

Arguments

- **Model** (Model)

Model that all shells will be unsketched in

- **flag** (Flag)

Flag set on the shells that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the shells are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all shells flagged with flag in model m:

```
Shell.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

Shell object.

Return type

Shell

Example

To check if Shell property s.example is a parameter by using the Shell.GetParameter() method:

```
if (s.ViewParameters().GetParameter(s.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for shell with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The shell object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for shell s:

```
s.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for shell s:

```
s.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for shell. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for shell s:

```
s.Warning("My custom warning");
```

Warpage()

Description

Calculates the warpage for the shell

Arguments

No arguments

Returns

real

Return type

Number

Example

To calculate the warpage for shell s:

```
var warpage = s.Warpage();
```

WeightingFactors(s*[real]*, t*[real]*)

Description

Calculates the weighting factors for a point on the shell from the isoparametric coords.

Arguments

- **s** (real)

First isoparametric coordinate

- **t** (real)

Second isoparametric coordinate

Returns

Array of numbers containing weighting factors or null if not possible to calculate.

Return type

Number

Example

To calculate the weighting factors of isoparametric point (0.5, -0.5) on shell s:

```
var weights = s.WeightingFactors(0.5, -0.5);
```

Xrefs()

Description

Returns the cross references for this shell.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for shell s:

```
var xrefs = s.Xrefs();
```

---

toString()

Description

Creates a string containing the shell data in keyword format. Note that this contains the keyword header and the keyword cards. See also Shell.Keyword() and Shell.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for shell s in keyword format

```
var str = s.toString();
```

Slipring class

The Slipring class gives you access to seatbelt slipring cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- ExtractColour()
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)

- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

Slipring properties

Name	Type	Description
colour	Colour	The colour of the slipring
dc	real	Optional decay constant to allow a smooth transition between the static and dynamic friction coefficients.
direct	integer	Direction of belt movement
exists (read only)	logical	true if slipring exists, false if referred to but not defined.
fc	real	Coulomb dynamic friction coefficient
fcs	real	Coulomb static friction coefficient
funcid	integer	Function ID to determine friction coefficient
include	integer	The Include file number that the slipring is in.
k	real	Optional coefficient for determining the Coulomb friction coefficient related to angle alpha
label	integer	Slipring number. Also see the sbsrid property which is an alternative name for this.
lcnffd	integer	Loadcurve for Coulomb dynamic friction
lcnffs	integer	Loadcurve for Coulomb static friction
ltime	real	Slipring lockup time
model (read only)	integer	The Model number that the slipring is in.
onid	integer	Orientation Node number
sbid1	integer	Seatbelt number 1 (or Set Shell number if sbrnid is negative).
sbid2	integer	Seatbelt number 2 (or Set Shell number if sbrnid is negative).
sbrnid	integer	Node number (or Set Node number if negative)
sbsrid	integer	Slipring number. Also see the label property which is an alternative name for this.
shell_seatbelt (read only)	logical	true if slipring is used for shell (2D) seatbelt elements.
transparency	integer	The transparency of the slipring (0-100) 0% is opaque, 100% is transparent.

Detailed Description

The Slipring class allows you to create, modify, edit and manipulate seatbelt slipring cards. See the documentation below for more details.

Constructor

new Slipring(Model*[Model]*, sbsrid*[integer]*, sbid1*[integer]*, sbid2*[integer]*, sbrnid*[integer]*)

Description

Create a new Seatbelt Slipring object.

Arguments

- **Model** (Model)

Model that slipring will be created in

- **sbsrid** (integer)

Slipring number.

- **sbid1** (integer)

Seatbelt number 1

- **sbid2** (integer)

Seatbelt number 2

- **sbrnid** (integer)

Slipring Node number

Returns

Slipring object

Return type

Slipring

Example

To create a new seatbelt slipring in model m with label 100, seatbelts 10, 11 and node 20:

```
var a = new Slipring(m, 100, 10, 11, 20);
```

Details of functions

AssociateComment(Comment[*Comment*])

Description

Associates a comment with a slipring.

Arguments

- **Comment** (Comment)

Comment that will be attached to the slipring

Returns

No return value

Example

To associate comment c to the slipring s:

```
s.AssociateComment(c);
```

Blank()

Description

Blanks the slipring

Arguments

No arguments

Returns

No return value

Example

To blank slipring s:

```
s.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the sliprings in the model.

Arguments

- **Model** (Model)

Model that all sliprings will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the sliprings in model m:

```
Slipring.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged sliprings in the model.

Arguments

- **Model** (Model)

Model that all the flagged sliprings will be blanked in

- **flag** (Flag)

Flag set on the sliprings that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the sliprings in model m flagged with f:

```
Slipring.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the slipring is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if slipring s is blanked:

```
if (s.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse slipring s:

```
s.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the slipring.

Arguments

- **flag** (Flag)

Flag to clear on the slipring

Returns

No return value

Example

To clear flag f for slipring s:

```
s.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the slipring. The target include of the copied slipring can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

Slipring object

Return type

Slipring

Example

To copy slipring s into slipring z:

```
var z = s.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a slipring

Arguments

- **Model** (Model)

Model that the slipring will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

Slipring object (or null if not made).

Return type

Slipring

Example

To start creating a slipring in model m:

```
var s = Slipring.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a slipring.

Arguments

- **Comment** (Comment)

Comment that will be detached from the slipring

Returns

No return value

Example

To detach comment c from the slipring s:

```
s.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit slipring s:

```
s.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for slipring with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description

autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The slipring object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for slipring s:

```
s.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for slipring s:

```
s.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for slipring. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for slipring s:

```
s.Error("My custom error");
```

ExtractColour()

Description

Extracts the **actual** colour used for slipring.
By default in PRIMER many entities such as elements get their colour automatically from the part that they are in. PRIMER cycles through 13 default colours based on the label of the entity. In this case the slipring colour property will return the value Colour.PART instead of the actual colour. This method will return the actual colour which is used for drawing the slipring.

Arguments

No arguments

Returns

colour value (integer)

Return type

Number

Example

To return the colour used for drawing slipring s:

```
var colour = s.ExtractColour();
```

First(Model*[Model]*) [static]

Description

Returns the first slipring in the model.

Arguments

- **Model** (Model)

Model to get first slipring in

Returns

Slipring object (or null if there are no sliprings in the model).

Return type

Slipring

Example

To get the first slipring in model m:

```
var s = Slipring.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free slipring label in the model. Also see Slipring.LastFreeLabel(), Slipring.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free slipring label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

Slipring label.

Return type

Number

Example

To get the first free slipring label in model m:

```
var label = Slipring.FirstFreeLabel(m);
```

FlagAll(Model[*Model*], flag[*Flag*]) [static]

Description

Flags all of the sliprings in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all sliprings will be flagged in

- **flag** (Flag)

Flag to set on the sliprings

Returns

No return value

Example

To flag all of the sliprings with flag f in model m:

```
Slipring.FlagAll(m, f);
```

Flagged(flag[*Flag*])

Description

Checks if the slipring is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the slipring

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if slipring s has flag f set on it:

```
if (s.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each slipring in the model.
Note that ForEach has been designed to make looping over sliprings as fast as possible and so has some limitations.
Firstly, a single temporary Slipring object is created and on each function call it is updated with the current slipring data. This means that you should not try to store the Slipring object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new sliprings inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all sliprings are in

- **func** (function)

Function to call for each slipring

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the sliprings in model m:

```
Slipring.ForEach(m, test);
function test(s)
{
// s is Slipring object
}
```

To call function test for all of the sliprings in model m with optional object:

```
var data = { x:0, y:0 };
Slipring.ForEach(m, test, data);
function test(s, extra)
{
// s is Slipring object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Slipring objects or properties for all of the sliprings in a model in PRIMER. If the optional property argument is not given then an array of Slipring objects is returned. If the property argument is given, that property value for each slipring is returned in the array instead of a Slipring object

Arguments

- **Model** (Model)

Model to get sliprings from

- **property (optional)** (string)

Name for property to get for all sliprings in the model

Returns

Array of Slipring objects or properties

Return type

Array

Example

To make an array of Slipring objects for all of the sliprings in model m:

```
var a = Slipring.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each slipring in model m:

```
var a = Slipring.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a slipring.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the slipring s:

```
var comm_array = s.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of Slipring objects for all of the flagged sliprings in a model in PRIMER If the optional property argument is not given then an array of Slipring objects is returned. If the property argument is given, then that property value for each slipring is returned in the array instead of a Slipring object

Arguments

- **Model** (Model)

Model to get sliprings from

- **flag** (Flag)

Flag set on the sliprings that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged sliprings in the model

Returns

Array of Slipring objects or properties

Return type

Array

Example

To make an array of Slipring objects for all of the sliprings in model m flagged with f:

```
var s = Slipring.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the sliprings in model m flagged with f:

```
var a = Slipring.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the Slipring object for a slipring ID.

Arguments

- **Model** (Model)

Model to find the slipring in

- **number** (integer)

number of the slipring you want the Slipring object for

Returns

Slipring object (or null if slipring does not exist).

Return type

Slipring

Example

To get the Slipring object for slipring 100 in model m

```
var s = Slipring.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a Slipring property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the Slipring.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

slipring property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if Slipring property s.example is a parameter:

```
Options.property_parameter_names = true;
if (s.GetParameter(s.example) ) do_something...
Options.property_parameter_names = false;
```

To check if Slipring property s.example is a parameter by using the GetParameter method:

```
if (s.ViewParameters().GetParameter(s.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this slipring (*ELEMENT_SEATBELT_SLIPEROMETER) **Note that a carriage return is not added**. See also Slipring.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for slipring s:

```
var key = s.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the slipring. **Note that a carriage return is not added**. See also Slipring.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for slipring s:

```
var cards = s.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last slipring in the model.

Arguments

- **Model** (Model)

Model to get last slipring in

Returns

Slipring object (or null if there are no sliprings in the model).

Return type

Slipring

Example

To get the last slipring in model m:

```
var s = Slipring.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free slipring label in the model. Also see Slipring.FirstFreeLabel(), Slipring.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free slipring label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

Slipring label.

Return type

Number

Example

To get the last free slipring label in model m:

```
var label = Slipring.LastFreeLabel(m);
```

Next()

Description

Returns the next slipring in the model.

Arguments

No arguments

Returns

Slipring object (or null if there are no more sliprings in the model).

Return type

Slipring

Example

To get the slipring in model m after slipring s:

```
var s = s.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) slipring label in the model. Also see Slipring.FirstFreeLabel(), Slipring.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free slipring label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

Slipring label.

Return type

Number

Example

To get the next free slipring label in model m:

```
var label = Slipring.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a slipring.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only sliprings from that model can be picked. If the argument is a Flag then only sliprings that are flagged with *limit* can be selected. If omitted, or null, any sliprings from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

Slipring object (or null if not picked)

Return type

Slipring

Example

To pick a slipring from model m giving the prompt 'Pick slipring from screen':

```
var s = Slipring.Pick('Pick slipring from screen', m);
```

Previous()

Description

Returns the previous slipring in the model.

Arguments

No arguments

Returns

Slipring object (or null if there are no more sliprings in the model).

Return type

Slipring

Example

To get the slipring in model m before slipring s:

```
var s = s.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the sliprings in the model.

Arguments

- **Model** (Model)

Model that all sliprings will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the sliprings in model m, from 1000000:

```
Slipring.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged sliprings in the model.

Arguments

- **Model** (Model)

Model that all the flagged sliprings will be renumbered in

- **flag** (Flag)

Flag set on the sliprings that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the sliprings in model m flagged with f, from 1000000:

```
Slipring.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select sliprings using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting sliprings

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only sliprings from that model can be selected. If the argument is a Flag then only sliprings that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any sliprings can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of sliprings selected or null if menu cancelled

Return type

Number

Example

To select sliprings from model m, flagging those selected with flag f, giving the prompt 'Select sliprings':

```
Slipring.Select(f, 'Select sliprings', m);
```

To select sliprings, flagging those selected with flag f but limiting selection to sliprings flagged with flag l, giving the prompt 'Select sliprings':

```
Slipring.Select(f, 'Select sliprings', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the slipring.

Arguments

- **flag** (Flag)

Flag to set on the slipring

Returns

No return value

Example

To set flag f for slipring s:

```
s.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the slipring. The slipring will be sketched until you either call Slipring.Unsketch(), Slipring.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the slipring is sketched. If omitted redraw is true. If you want to sketch several sliprings and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch slipring s:

```
s.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged sliprings in the model. The sliprings will be sketched until you either call Slipring.Unsketch(), Slipring.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged sliprings will be sketched in

- **flag** (Flag)

Flag set on the sliprings that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the sliprings are sketched. If omitted redraw is true. If you want to sketch flagged sliprings several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all sliprings flagged with flag in model m:

```
Slipring.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of sliprings in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing sliprings should be counted. If false or omitted referenced but undefined sliprings will also be included in the total.

Returns

number of sliprings

Return type

Number

Example

To get the total number of sliprings in model m:

```
var total = Slipring.Total(m);
```

Unblank()

Description

Unblanks the slipring

Arguments

No arguments

Returns

No return value

Example

To unblank slipring s:

```
s.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the sliprings in the model.

Arguments

- **Model** (Model)

Model that all sliprings will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the sliprings in model m:

```
Slipring.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged sliprings in the model.

Arguments

- **Model** (Model)

Model that the flagged sliprings will be unblanked in

- **flag** (Flag)

Flag set on the sliprings that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the sliprings in model m flagged with f:

```
Slipring.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the sliprings in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all sliprings will be unset in

- **flag** (Flag)

Flag to unset on the sliprings

Returns

No return value

Example

To unset the flag f on all the sliprings in model m:

```
Slipring.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the slipring.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the slipring is unsketched. If omitted redraw is true. If you want to unsketch several sliprings and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch slipring s:

```
s.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all sliprings.

Arguments

- **Model** (Model)

Model that all sliprings will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the sliprings are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all sliprings in model m:

```
Slipring.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged sliprings in the model.

Arguments

- **Model** (Model)

Model that all sliprings will be unsketched in

- **flag** (Flag)

Flag set on the sliprings that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the sliprings are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all sliprings flagged with flag in model m:

```
Slipring.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

Slipring object.

Return type

Slipring

Example

To check if Slipring property s.example is a parameter by using the Slipring.GetParameter() method:

```
if (s.ViewParameters().GetParameter(s.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for slipring with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The slipring object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for slipring s:

```
s.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for slipring s:

```
s.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for slipring. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for slipring s:

```
s.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this slipring.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for slipring s:

```
var xrefs = s.Xrefs();
```

---

toString()

Description

Creates a string containing the slipring data in keyword format. Note that this contains the keyword header and the keyword cards. See also Slipring.Keyword() and Slipring.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for slipring s in keyword format

```
var str = s.toString();
```

Solid class

The Solid class gives you access to solid cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- CoordsToIsoparametric(Model*[Model]*, x*[real]*, y*[real]*, z*[real]*, n1*[integer]*, n2*[integer]*, n3*[integer]*, n4*[integer]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- FindSolidInBox(Model*[Model]*, xmin*[real]*, xmax*[real]*, ymin*[real]*, ymax*[real]*, zmin*[real]*, zmax*[real]*, flag (optional)*[integer]*, excl (optional)*[integer]*, vis_only (optional)*[integer]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AspectRatio()
- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- ElemCut(Database cross section label*[integer]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- ExtractColour()
- Faces()
- Flagged(flag*[Flag]*)
- GetComments()
- GetNodeIDs()
- GetNodes()
- GetParameter(prop*[string]*)
- Jacobian()

- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- TetCollapse()
- TiedNodeCheck(Contact label*[integer]*, Flag*[Flag]*, Option1*[integer]*, Option2*[integer]*)
- Timestep()
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Volume()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Warpage()
- Xrefs()
- toString()

Solid constants

Name	Description
Solid.EDGE_1	Edge 1 of solid
Solid.EDGE_10	Edge 10 of solid
Solid.EDGE_11	Edge 11 of solid
Solid.EDGE_12	Edge 12 of solid
Solid.EDGE_2	Edge 2 of solid
Solid.EDGE_3	Edge 3 of solid
Solid.EDGE_4	Edge 4 of solid
Solid.EDGE_5	Edge 5 of solid
Solid.EDGE_6	Edge 6 of solid
Solid.EDGE_7	Edge 7 of solid
Solid.EDGE_8	Edge 8 of solid
Solid.EDGE_9	Edge 9 of solid
Solid.FACE_1	Face 1 of solid
Solid.FACE_2	Face 2 of solid
Solid.FACE_3	Face 3 of solid
Solid.FACE_4	Face 4 of solid
Solid.FACE_5	Face 5 of solid
Solid.FACE_6	Face 6 of solid

Solid properties

Name	Type	Description
a1	real	x component of material direction a
a2	real	y component of material direction a
a3	real	z component of material direction a
colour	Colour	The colour of the solid

d1	real	x component of material in-plane vector
d2	real	y component of material in-plane vector
d3	real	z component of material in-plane vector
dof	logical	If DOF option is set. Can be true or false
edges	constant	Bitwise code of Solid.EDGE_1, Solid.EDGE_2, Solid.EDGE_3, Solid.EDGE_4, Solid.EDGE_5, Solid.EDGE_6, Solid.EDGE_7, Solid.EDGE_8, Solid.EDGE_9, Solid.EDGE_10, Solid.EDGE_11 and Solid.EDGE_12 representing which edges of the solid are free faces
eid	integer	Solid number. Also see the label property which is an alternative name for this.
exists (read only)	logical	true if solid exists, false if referred to but not defined.
faces	constant	Bitwise code of Solid.FACE_1, Solid.FACE_2, Solid.FACE_3, Solid.FACE_4, Solid.FACE_5 and Solid.FACE_6 representing which faces of the solid are internal faces. Note that this is calculated from the solids that are currently visible so blanking solids will affect this property once graphics have been updated.
h20	logical	If _H20 option is set. Can be true or false
h27	logical	If _H27 option is set. Can be true or false
h64	logical	If _H64 option is set. Can be true or false
h8toh20	logical	If _H8TOH20 option is set. Can be true or false
h8toh27	logical	If _H8TOH27 option is set. Can be true or false
h8toh64	logical	If _H8TOH64 option is set. Can be true or false
include	integer	The Include file number that the solid is in.
label	integer	Solid number. Also see the eid property which is an alternative name for this.
model (read only)	integer	The Model number that the solid is in.
n1	integer	Node number 1
n10	integer	Node number 10
n11	integer	Node number 11
n12	integer	Node number 12
n13	integer	Node number 13
n14	integer	Node number 14
n15	integer	Node number 15
n16	integer	Node number 16
n17	integer	Node number 17
n18	integer	Node number 18
n19	integer	Node number 19
n2	integer	Node number 2
n20	integer	Node number 20
n21	integer	Node number 21
n22	integer	Node number 22
n23	integer	Node number 23
n24	integer	Node number 24

n25	integer	Node number 25
n26	integer	Node number 26
n27	integer	Node number 27
n28	integer	Node number 28
n29	integer	Node number 29
n3	integer	Node number 3
n30	integer	Node number 30
n31	integer	Node number 31
n32	integer	Node number 32
n33	integer	Node number 33
n34	integer	Node number 34
n35	integer	Node number 35
n36	integer	Node number 36
n37	integer	Node number 37
n38	integer	Node number 38
n39	integer	Node number 39
n4	integer	Node number 4
n40	integer	Node number 40
n41	integer	Node number 41
n42	integer	Node number 42
n43	integer	Node number 43
n44	integer	Node number 44
n45	integer	Node number 45
n46	integer	Node number 46
n47	integer	Node number 47
n48	integer	Node number 48
n49	integer	Node number 49
n5	integer	Node number 5
n50	integer	Node number 50
n51	integer	Node number 51
n52	integer	Node number 52
n53	integer	Node number 53
n54	integer	Node number 54
n55	integer	Node number 55
n56	integer	Node number 56
n57	integer	Node number 57
n58	integer	Node number 58
n59	integer	Node number 59
n6	integer	Node number 6

n60	integer	Node number 60
n61	integer	Node number 61
n62	integer	Node number 62
n63	integer	Node number 63
n64	integer	Node number 64
n7	integer	Node number 7
n8	integer	Node number 8
n9	integer	Node number 9
nodes (read only)	integer	Number of nodes solid has
ns1	integer	Scalar Node number 1
ns2	integer	Scalar Node number 2
ns3	integer	Scalar Node number 3
ns4	integer	Scalar Node number 4
ns5	integer	Scalar Node number 5
ns6	integer	Scalar Node number 6
ns7	integer	Scalar Node number 7
ns8	integer	Scalar Node number 8
ortho	logical	If _ORTHO option is set. Can be true or false
p21	logical	If _P21 option is set. Can be true or false
p40	logical	If _P40 option is set. Can be true or false
pid	integer	Part number
t15	logical	If _T15 option is set. Can be true or false
t20	logical	If _T20 option is set. Can be true or false
tet4totet10	logical	If _TET4TOTET10 option is set. Can be true or false
transparency	integer	The transparency of the solid (0-100) 0% is opaque, 100% is transparent.

Detailed Description

The Solid class allows you to create, modify, edit and manipulate solid cards. See the documentation below for more details.

Constructor

new Solid(Model*[Model]*, options *[object]*)

Description

Create a new Solid object. If you are creating a 4 noded solid either only give 4 nodes or give 8 nodes but make nodes 4 to 8 the same number. If you are creating a 6 noded solid either only give 6 nodes or give 8 nodes but make nodes 5 and 6 the same number and nodes 7 and 8 the same number.

Arguments

- **Model** (Model)

Model that solid will be created in

- **options** (object)

Options for creating the solid

Object has the following properties:

Name	Type	Description
eid	integer	Solid number
nodes	Array of numbers	Array of Node IDs for the solid. At least 4 nodes must be given
pid	integer	Part number

Returns

Solid object

Return type

Solid

Example

To create a new solid in model m with label 100, part 10 and nodes 1, 2, 3, 4:

```
var s = new Solid(m, {eid: 100, pid: 10, nodes: [1,2,3,4]} );
```

new Solid(Model*[Model]*, eid*[integer]*, pid*[integer]*, n1*[integer]*, n2*[integer]*, n3*[integer]*, n4*[integer]*, n5 (optional)*[integer]*, n6 (optional)*[integer]*, n7 (optional)*[integer]*, n8 (optional)*[integer]*) [deprecated]

This function is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Create a new Solid object. Use either 4, 6 or 8 nodes when creating a new solid. If you are creating a 4 noded solid either only give 4 nodes or give 8 nodes but make nodes 4 to 8 the same number. If you are creating a 6 noded solid either only give 6 nodes or give 8 nodes but make nodes 5 and 6 the same number and nodes 7 and 8 the same number.

Arguments

- **Model** (Model)

Model that solid will be created in

- **eid** (integer)

Solid number

- **pid** (integer)

Part number

- **n1** (integer)

Node number 1 or array containing all nodes (in which case other no other arguement has to be given after this)

- **n2** (integer)

Node number 2

- **n3** (integer)

Node number 3

- **n4** (integer)

Node number 4

- **n5 (optional)** (integer)

Node number 5

- **n6 (optional)** (integer)

Node number 6

- **n7 (optional)** (integer)

Node number 7

- **n8 (optional)** (integer)

Node number 8

Returns

Solid object

Return type

Solid

Example

To create a new solid in model m with label 100, part 10 and nodes 1, 2, 3, 4, 5, 6, 7, 8:

```
var s = new Solid(m, 100, 10, 1, 2, 3, 4, 5, 6, 7, 8);
```

Details of functions

AspectRatio()

Description

Calculates the aspect ratio for the solid

Arguments

No arguments

Returns

real

Return type

Number

Example

To calculate the aspect ratio for solid s:

```
var ratio = s.AspectRatio();
```

AssociateComment(Comment[*Comment*])

Description

Associates a comment with a solid.

Arguments

- **Comment** (Comment)

Comment that will be attached to the solid

Returns

No return value

Example

To associate comment c to the solid s:

```
s.AssociateComment(c);
```

Blank()

Description

Blanks the solid

Arguments

No arguments

Returns

No return value

Example

To blank solid s:

```
s.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the solids in the model.

Arguments

- **Model** (Model)

Model that all solids will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the solids in model m:

```
Solid.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged solids in the model.

Arguments

- **Model** (Model)

Model that all the flagged solids will be blanked in

- **flag** (Flag)

Flag set on the solids that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the solids in model m flagged with f:

```
Solid.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the solid is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if solid s is blanked:

```
if (s.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse solid s:

```
s.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the solid.

Arguments

- **flag** (Flag)

Flag to clear on the solid

Returns

No return value

Example

To clear flag f for solid s:

```
s.ClearFlag(f);
```

CoordsToIsoparametric(Model*[Model]*, x*[real]*, y*[real]*, z*[real]*, n1*[integer]*, n2*[integer]*, n3*[integer]*, n4*[integer]*) [static]

Description

Calculates the isoparametric coordinates for a point on 3 or 4 noded segment

Arguments

- **Model** (Model)

Model designated model

- **x** (real)

X coordinate of point

- **y** (real)

Y coordinate of point

- **z** (real)

Z coordinate of point

- **n1** (integer)

node 1 of segment

- **n2** (integer)

node 2 of segment

- **n3** (integer)

node 3 of segment

- **n4** (integer)

node 4 of segment

Returns

Array containing s and t isoparametric coordinates and the distance the point is from the segment If it is not possible to calculate the isoparametric coordinates null is returned.

Return type

Array

Example

To calculate the isoparametric coordinates of point (100, 100, 20) on segment defined by nodes 11,12,13,14:

```
var isocoords = Solid.CoordsToIsoparametric(100, 100, 20, 11, 12, 13, 14);
```

Copy(range (optional)*[boolean]*)

Description

Copies the solid. The target include of the copied solid can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

Solid object

Return type

Solid

Example

To copy solid s into solid z:

```
var z = s.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a solid

Arguments

- **Model** (Model)

Model that the solid will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

Solid object (or null if not made).

Return type

Solid

Example

To start creating a solid in model m:

```
var s = Solid.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a solid.

Arguments

- **Comment** (Comment)

Comment that will be detached from the solid

Returns

No return value

Example

To detach comment c from the solid s:

```
s.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit solid s:

```
s.Edit();
```

ElemCut(Database cross section label*[integer]*)

Description

Returns coordinates of the intersections between a solid and a database cross section.

Arguments

- **Database cross section label** (integer)

The label of the database cross section.

Returns

Object with the following properties:

Name	Type	Description
face1	Array of reals	An array containing the x1,y1,z1,x2,y2,z2 coordinates of the cut line on the face 1. Null if no cut on this face.
face2	Array of reals	An array containing the x1,y1,z1,x2,y2,z2 coordinates of the cut line on the face 2. Null if no cut on this face.
face3	Array of reals	An array containing the x1,y1,z1,x2,y2,z2 coordinates of the cut line on the face 3. Null if no cut on this face.
face4	Array of reals	An array containing the x1,y1,z1,x2,y2,z2 coordinates of the cut line on the face 4. Null if no cut on this face.
face5	Array of reals	An array containing the x1,y1,z1,x2,y2,z2 coordinates of the cut line on the face 5. Null if no cut on this face.
face6	Array of reals	An array containing the x1,y1,z1,x2,y2,z2 coordinates of the cut line on the face 6. Null if no cut on this face.

Return type

object

Example

To see if the database cross section 200 cuts solid s and at which points it cuts face 3 of the solid:

```
var data = s.ElemCut(200);
var face = data.face3;
if(face)
{
   var point1_x = face[0];
   var point1_y = face[1];
   var point1_z = face[2];
   var point2_x = face[3];
   var point2_y = face[4];
   var point2_z = face[5];
}
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for solid with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The solid object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for solid s:

```
s.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for solid s:

```
s.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for solid. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for solid s:

```
s.Error("My custom error");
```

ExtractColour()

Description

Extracts the **actual** colour used for solid.
By default in PRIMER many entities such as elements get their colour automatically from the part that they are in. PRIMER cycles through 13 default colours based on the label of the entity. In this case the solid colour property will return the value Colour.PART instead of the actual colour. This method will return the actual colour which is used for drawing the solid.

Arguments

No arguments

Returns

colour value (integer)

Return type

Number

Example

To return the colour used for drawing solid s:

```
var colour = s.ExtractColour();
```

Faces()

Description

Returns an array of node arrays defining the faces of the solid. You must ensure that the solid exists before calling this method.

Arguments

No arguments

Returns

Array of Node arrays

Return type

Array

Example

To print the IDs of the nodes that define each face of solid s:

```
var faces = s.Faces();
for(var i=0; i<faces.length; i++)
{
    var nodes = faces[i];
    var msg = 'Nodes for face ${i}: ';
    for(var j=0; j<nodes.length; j++)
    {
        var n = nodes[j];
        if(j == 0) msg = msg + '${j}';
        else       msg = msg + ', ${j}';
    }
    Message(msg);
}
```

FindSolidInBox(Model*[Model]*, xmin*[real]*, xmax*[real]*, ymin*[real]*, ymax*[real]*, zmin*[real]*, zmax*[real]*, flag (optional)*[integer]*, excl (optional)*[integer]*, vis_only (optional)*[integer]*) [static]

Description

Returns an array of Solid objects for the solids within a box. Please note this function provides a list of all solids that could potentially be in the box (using computationally cheap bounding box comparison) it is not a rigorous test of whether the solid is actually in the box. This may include solids that are ostensibly outside box. The user should apply their own test. (this function is intended to provide an upper bound of elems to test) Setting the "excl" flag will require that the solid is fully contained but this may not capture all the solids you want to process.

Arguments

- **Model** (Model)

Model designated model

- **xmin** (real)

Minimum bound in global x

- **xmax** (real)

Maximum bound in global x

- **ymin** (real)

Minimum bound in global y

- **ymax** (real)

Maximum bound in global y

- **zmin** (real)

Minimum bound in global z

- **zmax** (real)

Maximum bound in global z

- **flag (optional)** (integer)

Optional flag to restrict solids considered, if 0 all solids considered

- **excl (optional)** (integer)

Optional flag (0) Apply inclusive selection (1) Apply exclusive selection inclusive selection means elements intersect

box exclusive selection means elements contained in box

- **vis_only (optional)** (integer)

Optional flag to consider visible elements only (1), if (0) all elements considered

Returns

Array of Solid objects

Return type

Array

Example

To get an array of Solid objects for flagged solids within defined box (inclusive selection)

```
var s = Solid.FindSolidInBox(m, xmin, xmax, ymin, ymax, zmin, zmax, flag, 0, 0);
```

First(Model*[Model]*) [static]

Description

Returns the first solid in the model.

Arguments

- **Model** (Model)

Model to get first solid in

Returns

Solid object (or null if there are no solids in the model).

Return type

Solid

Example

To get the first solid in model m:

```
var s = Solid.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free solid label in the model. Also see Solid.LastFreeLabel(), Solid.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free solid label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

Solid label.

Return type

Number

Example

To get the first free solid label in model m:

```
var label = Solid.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the solids in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all solids will be flagged in

- **flag** (Flag)

Flag to set on the solids

Returns

No return value

Example

To flag all of the solids with flag f in model m:

```
Solid.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the solid is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the solid

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if solid s has flag f set on it:

```
if (s.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each solid in the model.
Note that ForEach has been designed to make looping over solids as fast as possible and so has some limitations. Firstly, a single temporary Solid object is created and on each function call it is updated with the current solid data. This means that you should not try to store the Solid object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new solids inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all solids are in

- **func** (function)

Function to call for each solid

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the solids in model m:

```
Solid.ForEach(m, test);
function test(s)
{
// s is Solid object
}
```

To call function test for all of the solids in model m with optional object:

```
var data = { x:0, y:0 };
Solid.ForEach(m, test, data);
function test(s, extra)
{
// s is Solid object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Solid objects or properties for all of the solids in a model in PRIMER. If the optional property argument is not given then an array of Solid objects is returned. If the property argument is given, that property value for each solid is returned in the array instead of a Solid object

Arguments

- **Model** (Model)

Model to get solids from

- **property (optional)** (string)

Name for property to get for all solids in the model

Returns

Array of Solid objects or properties

Return type

Array

Example

To make an array of Solid objects for all of the solids in model m:

```
var a = Solid.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each solid in model m:

```
var a = Solid.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a solid.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the solid s:

```
var comm_array = s.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of Solid objects for all of the flagged solids in a model in PRIMER If the optional property argument is not given then an array of Solid objects is returned. If the property argument is given, then that property value for each solid is returned in the array instead of a Solid object

Arguments

- **Model** (Model)

Model to get solids from

- **flag** (Flag)

Flag set on the solids that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged solids in the model

Returns

Array of Solid objects or properties

Return type

Array

Example

To make an array of Solid objects for all of the solids in model m flagged with f:

```
var s = Solid.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the solids in model m flagged with f:

```
var a = Solid.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the Solid object for a solid ID.

Arguments

- **Model** (Model)

Model to find the solid in

- **number** (integer)

number of the solid you want the Solid object for

Returns

Solid object (or null if solid does not exist).

Return type

Solid

Example

To get the Solid object for solid 100 in model m

```
var s = Solid.GetFromID(m, 100);
```

GetNodeIDs()

Description

Returns the labels of the nodes on the solid as an array. See also Solid.GetNodes()

Arguments

No arguments

Returns

Array of node labels (integers)

Return type

Number

Example

To return the node labels of solid s as an array

```
var nodes = s.GetNodeIDs();
```

GetNodes()

Description

Returns the nodes on the solid as an array of Node objects. See also Solid.GetNodeIDs()

Arguments

No arguments

Returns

Array of Node objects

Return type

Array

Example

To return the nodes of solid s as an array

```
var nodes = s.GetNodes();
```

GetParameter(prop*[string]*)

Description

Checks if a Solid property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the Solid.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

solid property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if Solid property s.example is a parameter:

```
Options.property_parameter_names = true;
if (s.GetParameter(s.example) ) do_something...
Options.property_parameter_names = false;
```

To check if Solid property s.example is a parameter by using the GetParameter method:

```
if (s.ViewParameters().GetParameter(s.example) ) do_something...
```

Jacobian()

Description

Calculates the jacobian for the solid

Arguments

No arguments

Returns

real

Return type

Number

Example

To calculate the jacobian for solid s:

```
var jacobian = s.Jacobian();
```

Keyword()

Description

Returns the keyword for this solid (*SOLID, *SOLID_SCALAR or *SOLID_SCALAR_VALUE). **Note that a carriage return is not added**. See also Solid.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for solid s:

```
var key = s.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the solid. **Note that a carriage return is not added**. See also Solid.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for solid s:

```
var cards = s.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last solid in the model.

Arguments

- **Model** (Model)

Model to get last solid in

Returns

Solid object (or null if there are no solids in the model).

Return type

Solid

Example

To get the last solid in model m:

```
var s = Solid.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free solid label in the model. Also see Solid.FirstFreeLabel(), Solid.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free solid label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

Solid label.

Return type

Number

Example

To get the last free solid label in model m:

```
var label = Solid.LastFreeLabel(m);
```

Next()

Description

Returns the next solid in the model.

Arguments

No arguments

Returns

Solid object (or null if there are no more solids in the model).

Return type

Solid

Example

To get the solid in model m after solid s:

```
var s = s.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) solid label in the model. Also see Solid.FirstFreeLabel(), Solid.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free solid label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

Solid label.

Return type

Number

Example

To get the next free solid label in model m:

```
var label = Solid.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a solid.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only solids from that model can be picked. If the argument is a Flag then only solids that are flagged with *limit* can be selected. If omitted, or null, any solids from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

Solid object (or null if not picked)

Return type

Solid

Example

To pick a solid from model m giving the prompt 'Pick solid from screen':

```
var s = Solid.Pick('Pick solid from screen', m);
```

Previous()

Description

Returns the previous solid in the model.

Arguments

No arguments

Returns

Solid object (or null if there are no more solids in the model).

Return type

Solid

Example

To get the solid in model m before solid s:

```
var s = s.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the solids in the model.

Arguments

- **Model** (Model)

Model that all solids will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the solids in model m, from 1000000:

```
Solid.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged solids in the model.

Arguments

- **Model** (Model)

Model that all the flagged solids will be renumbered in

- **flag** (Flag)

Flag set on the solids that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the solids in model m flagged with f, from 1000000:

```
Solid.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select solids using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting solids

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only solids from that model can be selected. If the argument is a Flag then only solids that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any solids can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of solids selected or null if menu cancelled

Return type

Number

Example

To select solids from model m, flagging those selected with flag f, giving the prompt 'Select solids':

```
Solid.Select(f, 'Select solids', m);
```

To select solids, flagging those selected with flag f but limiting selection to solids flagged with flag l, giving the prompt 'Select solids':

```
Solid.Select(f, 'Select solids', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the solid.

Arguments

- **flag** (Flag)

Flag to set on the solid

Returns

No return value

Example

To set flag f for solid s:

```
s.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the solid. The solid will be sketched until you either call Solid.Unsketch(), Solid.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the solid is sketched. If omitted redraw is true. If you want to sketch several solids and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch solid s:

```
s.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged solids in the model. The solids will be sketched until you either call Solid.Unsketch(), Solid.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged solids will be sketched in

- **flag** (Flag)

Flag set on the solids that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the solids are sketched. If omitted redraw is true. If you want to sketch flagged solids several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all solids flagged with flag in model m:

```
Solid.SketchFlagged(m, flag);
```

TetCollapse()

Description

Calculates the tetrahedral collapse for the solid

Arguments

No arguments

Returns

real

Return type

Number

Example

To calculate the tet collapse for solid s:

```
var tet collapse = s.TetCollapse();
```

TiedNodeCheck(Contact label*[integer]*, Flag*[Flag]*, Option1*[integer]*, Option2*[integer]*)

Description

Checks if nodes of solid are tied by contact or directly attached (non-zero option1).

Arguments

- **Contact label** (integer)

The label of the tied contact. If zero the tied contact is found for the solid by reverse lookup.

- **Flag** (Flag)

flag bit

- **Option1** (integer)

Directly tied node (logical OR) 0:NONE 1:NRB/C_EXNO 2:BEAM 4:SHELL 8:SOLID 16:TSHELL

- **Option2** (integer)

0:No action 1:report error if directly attached node (acc. option1) also captured by contact

Returns

string

Return type

String

Example

To check if all nodes of solid h are tied by contact 200 or attach directly to constraint or shell:

```
var message = h.TiedNodeCheck(200, flag, 1|4, 1)
```

Timestep()

Description

Calculates the timestep for the solid

Arguments

No arguments

Returns

real

Return type

Number

Example

To calculate the timestep for solid s:

```
var timestep = s.Timestep();
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of solids in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing solids should be counted. If false or omitted referenced but undefined solids will also be included in the total.

Returns

number of solids

Return type

Number

Example

To get the total number of solids in model m:

```
var total = Solid.Total(m);
```

Unblank()

Description

Unblanks the solid

Arguments

No arguments

Returns

No return value

Example

To unblank solid s:

```
s.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the solids in the model.

Arguments

- **Model** (Model)

Model that all solids will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the solids in model m:

```
Solid.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged solids in the model.

Arguments

- **Model** (Model)

Model that the flagged solids will be unblanked in

- **flag** (Flag)

Flag set on the solids that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the solids in model m flagged with f:

```
Solid.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the solids in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all solids will be unset in

- **flag** (Flag)

Flag to unset on the solids

Returns

No return value

Example

To unset the flag f on all the solids in model m:

```
Solid.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the solid.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the solid is unsketched. If omitted redraw is true. If you want to unsketch several solids and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch solid s:

```
s.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all solids.

Arguments

- **Model** (Model)

Model that all solids will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the solids are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all solids in model m:

```
Solid.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged solids in the model.

Arguments

- **Model** (Model)

Model that all solids will be unsketched in

- **flag** (Flag)

Flag set on the solids that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the solids are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all solids flagged with flag in model m:

```
Solid.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

Solid object.

Return type

Solid

Example

To check if Solid property s.example is a parameter by using the Solid.GetParameter() method:

```
if (s.ViewParameters().GetParameter(s.example) ) do_something...
```

Volume()

Description

Calculates the volume for the solid

Arguments

No arguments

Returns

real

Return type

Number

Example

To calculate the volume for solid s:

```
var volume = s.Volume();
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for solid with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The solid object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for solid s:

```
s.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for solid s:

```
s.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for solid. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for solid s:

```
s.Warning("My custom warning");
```

Warpage()

Description

Calculates the warpage for the solid

Arguments

No arguments

Returns

real

Return type

Number

Example

To calculate the warpage for solid s:

```
var warpage = s.Warpage();
```

Xrefs()

Description

Returns the cross references for this solid.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for solid s:

```
var xrefs = s.Xrefs();
```

---

toString()

Description

Creates a string containing the solid data in keyword format. Note that this contains the keyword header and the keyword cards. See also Solid.Keyword() and Solid.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for solid s in keyword format

```
var str = s.toString();
```

Sph class

The Sph class gives you access to Element SPH cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- ExtractColour()
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)

- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

Sph properties

Name	Type	Description
colour	Colour	The colour of the sph element.
exists (read only)	logical	true if sph exists, false if referred to but not defined.
include	integer	The Include file number that the sph is in.
mass	real	Mass value.
model (read only)	integer	The Model number that the sph is in.
nid	integer	Node ID.
pid	integer	Part ID to which this element belongs.
transparency	integer	The transparency of the sph (0-100) 0% is opaque, 100% is transparent.

Detailed Description

The Sph class allows you to create, modify, edit and manipulate SPH cards. See the documentation below for more details.

Constructor

new Sph(Model*[Model]*, nid*[integer]*, pid*[integer]*, mass*[real]*)

Description

Create a new object.

Arguments

- **Model** (Model)

Model that sph will be created in

- **nid** (integer)

Node ID and Element ID are the same for the SPH option.

- **pid** (integer)

Part ID to which this element belongs.

- **mass** (real)

Mass value.

Returns

Sph object

Return type

Sph

Example

To create a new sph element in model m with nid = 100, pid = 400, mass = 0.9:

```
var s = new Sph(m, 100, 400, 0.9);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a sph.

Arguments

- **Comment** (Comment)

Comment that will be attached to the sph

Returns

No return value

Example

To associate comment c to the sph s:

```
s.AssociateComment(c);
```

Blank()

Description

Blanks the sph

Arguments

No arguments

Returns

No return value

Example

To blank sph s:

```
s.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the sphs in the model.

Arguments

- **Model** (Model)

Model that all sphs will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the sphs in model m:

```
Sph.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged sphs in the model.

Arguments

- **Model** (Model)

Model that all the flagged sphs will be blanked in

- **flag** (Flag)

Flag set on the sphs that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the sphs in model m flagged with f:

```
Sph.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the sph is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if sph s is blanked:

```
if (s.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse sph s:

```
s.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the sph.

Arguments

- **flag** (Flag)

Flag to clear on the sph

Returns

No return value

Example

To clear flag f for sph s:

```
s.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the sph. The target include of the copied sph can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

Sph object

Return type

Sph

Example

To copy sph s into sph z:

```
var z = s.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a sph

Arguments

- **Model** (Model)

Model that the sph will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

Sph object (or null if not made).

Return type

Sph

Example

To start creating a sph in model m:

```
var s = Sph.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a sph.

Arguments

- **Comment** (Comment)

Comment that will be detached from the sph

Returns

No return value

Example

To detach comment c from the sph s:

```
s.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit sph s:

```
s.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for sph with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The sph object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for sph s:

```
s.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for sph s:

```
s.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for sph. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for sph s:

```
s.Error("My custom error");
```

ExtractColour()

Description

Extracts the **actual** colour used for sph.
By default in PRIMER many entities such as elements get their colour automatically from the part that they are in. PRIMER cycles through 13 default colours based on the label of the entity. In this case the sph colour property will return the value Colour.PART instead of the actual colour. This method will return the actual colour which is used for drawing the sph.

Arguments

No arguments

Returns

colour value (integer)

Return type

Number

Example

To return the colour used for drawing sph s:

```
var colour = s.ExtractColour();
```

First(Model[*Model*]) [static]

Description

Returns the first sph in the model.

Arguments

- **Model** (Model)

Model to get first sph in

Returns

Sph object (or null if there are no sphs in the model).

Return type

Sph

Example

To get the first sph in model m:

```
var s = Sph.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free sph label in the model. Also see Sph.LastFreeLabel(), Sph.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free sph label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

Sph label.

Return type

Number

Example

To get the first free sph label in model m:

```
var label = Sph.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the sphs in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all sphs will be flagged in

- **flag** (Flag)

Flag to set on the sphs

Returns

No return value

Example

To flag all of the sphs with flag f in model m:

```
Sph.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the sph is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the sph

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if sph s has flag f set on it:

```
if (s.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each sph in the model.
Note that ForEach has been designed to make looping over sphs as fast as possible and so has some limitations. Firstly, a single temporary Sph object is created and on each function call it is updated with the current sph data. This means that you should not try to store the Sph object for later use (e.g. in an array) as it is temporary. Secondly, you cannot create new sphs inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all sphs are in

- **func** (function)

Function to call for each sph

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the sphs in model m:

```
Sph.ForEach(m, test);
function test(s)
{
// s is Sph object
}
```

To call function test for all of the sphs in model m with optional object:

```
var data = { x:0, y:0 };
Sph.ForEach(m, test, data);
function test(s, extra)
{
// s is Sph object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Sph objects or properties for all of the sphs in a model in PRIMER. If the optional property argument is not given then an array of Sph objects is returned. If the property argument is given, that property value for each sph is returned in the array instead of a Sph object

Arguments

- **Model** (Model)

Model to get sphs from

- **property (optional)** (string)

Name for property to get for all sphs in the model

Returns

Array of Sph objects or properties

Return type

Array

Example

To make an array of Sph objects for all of the sphs in model m:

```
var a = Sph.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each sph in model m:

```
var a = Sph.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a sph.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the sph s:

```
var comm_array = s.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of Sph objects for all of the flagged sphs in a model in PRIMER If the optional property argument is not given then an array of Sph objects is returned. If the property argument is given, then that property value for each sph is returned in the array instead of a Sph object

Arguments

- **Model** (Model)

Model to get sphs from

- **flag** (Flag)

Flag set on the sphs that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged sphs in the model

Returns

Array of Sph objects or properties

Return type

Array

Example

To make an array of Sph objects for all of the sphs in model m flagged with f:

```
var s = Sph.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the sphs in model m flagged with f:

```
var a = Sph.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the Sph object for a sph ID.

Arguments

- **Model** (Model)

Model to find the sph in

- **number** (integer)

number of the sph you want the Sph object for

Returns

Sph object (or null if sph does not exist).

Return type

Sph

Example

To get the Sph object for sph 100 in model m

```
var s = Sph.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a Sph property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the Sph.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

sph property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if Sph property s.example is a parameter:

```
Options.property_parameter_names = true;
if (s.GetParameter(s.example) ) do_something...
Options.property_parameter_names = false;
```

To check if Sph property s.example is a parameter by using the GetParameter method:

```
if (s.ViewParameters().GetParameter(s.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this sph (*ELEMENT_SPH) **Note that a carriage return is not added**. See also Sph.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for sph s:

```
var key = s.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the sph. **Note that a carriage return is not added**. See also Sph.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for sph s:

```
var cards = s.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last sph in the model.

Arguments

- **Model** (Model)

Model to get last sph in

Returns

Sph object (or null if there are no sphs in the model).

Return type

Sph

Example

To get the last sph in model m:

```
var s = Sph.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free sph label in the model. Also see Sph.FirstFreeLabel(), Sph.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free sph label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

Sph label.

Return type

Number

Example

To get the last free sph label in model m:

```
var label = Sph.LastFreeLabel(m);
```

Next()

Description

Returns the next sph in the model.

Arguments

No arguments

Returns

Sph object (or null if there are no more sphs in the model).

Return type

Sph

Example

To get the sph in model m after sph s:

```
var s = s.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) sph label in the model. Also see Sph.FirstFreeLabel(), Sph.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free sph label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

Sph label.

Return type

Number

Example

To get the next free sph label in model m:

```
var label = Sph.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a sph.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only sphs from that model can be picked. If the argument is a Flag then only sphs that are flagged with *limit* can be selected. If omitted, or null, any sphs from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled ’Cancel’ which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted ’Cancel’ will be used.

Returns

Sph object (or null if not picked)

Return type

Sph

Example

To pick a sph from model m giving the prompt ’Pick sph from screen’:

```
var s = Sph.Pick(’Pick sph from screen’, m);
```

Previous()

Description

Returns the previous sph in the model.

Arguments

No arguments

Returns

Sph object (or null if there are no more sphs in the model).

Return type

Sph

Example

To get the sph in model m before sph s:

```
var s = s.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the sphs in the model.

Arguments

- **Model** (Model)

Model that all sphs will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the sphs in model m, from 1000000:

```
Sph.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged sphs in the model.

Arguments

- **Model** (Model)

Model that all the flagged sphs will be renumbered in

- **flag** (Flag)

Flag set on the sphs that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the sphs in model m flagged with f, from 1000000:

```
Sph.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select sphs using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting sphs

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only sphs from that model can be selected. If the argument is a Flag then only sphs that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any sphs can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of sphs selected or null if menu cancelled

Return type

Number

Example

To select sphs from model m, flagging those selected with flag f, giving the prompt 'Select sphs':

```
Sph.Select(f, 'Select sphs', m);
```

To select sphs, flagging those selected with flag f but limiting selection to sphs flagged with flag l, giving the prompt 'Select sphs':

```
Sph.Select(f, 'Select sphs', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the sph.

Arguments

- **flag** (Flag)

Flag to set on the sph

Returns

No return value

Example

To set flag f for sph s:

```
s.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the sph. The sph will be sketched until you either call Sph.Unsketch(), Sph.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the sph is sketched. If omitted redraw is true. If you want to sketch several sphs and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch sph s:

```
s.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged sphs in the model. The sphs will be sketched until you either call Sph.Unsketch(), Sph.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged sphs will be sketched in

- **flag** (Flag)

Flag set on the sphs that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the sphs are sketched. If omitted redraw is true. If you want to sketch flagged sphs several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all sphs flagged with flag in model m:

```
Sph.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of sphs in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing sphs should be counted. If false or omitted referenced but undefined sphs will also be included in the total.

Returns

number of sphs

Return type

Number

Example

To get the total number of sphs in model m:

```
var total = Sph.Total(m);
```

Unblank()

Description

Unblanks the sph

Arguments

No arguments

Returns

No return value

Example

To unblank sph s:

```
s.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the sphs in the model.

Arguments

- **Model** (Model)

Model that all sphs will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the sphs in model m:

```
Sph.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged sphs in the model.

Arguments

- **Model** (Model)

Model that the flagged sphs will be unblanked in

- **flag** (Flag)

Flag set on the sphs that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the sphs in model m flagged with f:

```
Sph.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the sphs in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all sphs will be unset in

- **flag** (Flag)

Flag to unset on the sphs

Returns

No return value

Example

To unset the flag f on all the sphs in model m:

```
Sph.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the sph.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the sph is unsketched. If omitted redraw is true. If you want to unsketch several sphs and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch sph s:

```
s.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all sphs.

Arguments

- **Model** (Model)

Model that all sphs will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the sphs are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all sphs in model m:

```
Sph.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged sphs in the model.

Arguments

- **Model** (Model)

Model that all sphs will be unsketched in

- **flag** (Flag)

Flag set on the sphs that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the sphs are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all sphs flagged with flag in model m:

```
Sph.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

Sph object.

Return type

Sph

Example

To check if Sph property s.example is a parameter by using the Sph.GetParameter() method:

```
if (s.ViewParameters().GetParameter(s.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for sph with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The sph object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for sph s:

```
s.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for sph s:

```
s.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for sph. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for sph s:

```
s.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this sph.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for sph s:

```
var xrefs = s.Xrefs();
```

---

toString()

Description

Creates a string containing the sph data in keyword format. Note that this contains the keyword header and the keyword cards. See also Sph.Keyword() and Sph.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for sph s in keyword format

```
var str = s.toString();
```

Tshell class

The Tshell class gives you access to thick shell cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- FindTshellInBox(Model*[Model]*, xmin*[real]*, xmax*[real]*, ymin*[real]*, ymax*[real]*, zmin*[real]*, zmax*[real]*, flag (optional)*[integer]*, excl (optional)*[integer]*, vis_only (optional)*[integer]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AspectRatio()
- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- ElemCut(Database cross section label*[integer]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- ExtractColour()
- Flagged(flag*[Flag]*)
- GetComments()
- GetCompositeData(ipt*[integer]*)
- GetNodeIDs()
- GetNodes()
- GetParameter(prop*[string]*)
- Jacobian()
- Keyword()
- KeywordCards()

- Next()
- Previous()
- RemoveCompositeData(ipt*[integer]*)
- SetCompositeData(ipt*[integer]*, mid*[integer]*, thick*[real]*, beta*[real]*)
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Timestep()
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Warpage()
- Xrefs()
- toString()

Tshell properties

Name	Type	Description
beta	logical	If BETA option is set.
beta_angle	real	Angle for BETA option.
colour	Colour	The colour of the thick shell
composite	logical	If COMPOSITE option is set. Can be true or false
eid	integer	Tshell number. Also see the label property which is an alternative name for this.
exists (read only)	logical	true if thick shell exists, false if referred to but not defined.
include	integer	The Include file number that the thick shell is in.
label	integer	Tshell number. Also see the eid property which is an alternative name for this.
model (read only)	integer	The Model number that the thick shell is in.
n1	integer	Node number 1
n2	integer	Node number 2
n3	integer	Node number 3
n4	integer	Node number 4
n5	integer	Node number 5
n6	integer	Node number 6
n7	integer	Node number 7
n8	integer	Node number 8
nip	logical	Number of integration points for composite thick shell
nodes (read only)	integer	Number of nodes thick shell has
pid	integer	Part number
transparency	integer	The transparency of the thick shell (0-100) 0% is opaque, 100% is transparent.

Detailed Description

The Tshell class allows you to create, modify, edit and manipulate thich shell cards. See the documentation below for more details.

Constructor

new Tshell(Model*[Model]*, eid*[integer]*, pid*[integer]*, n1*[integer]*, n2*[integer]*, n3*[integer]*, n4*[integer]*, n5*[integer]*, n6*[integer]*, n7 (optional)*[integer]*, n8 (optional)*[integer]*)

Description

Create a new Tshell object. Use either 6 or 8 nodes when creating a new thick shell.

Arguments

- **Model** (Model)

Model that thick shell will be created in

- **eid** (integer)

Tshell number

- **pid** (integer)

Part number

- **n1** (integer)

Node number 1

- **n2** (integer)

Node number 2

- **n3** (integer)

Node number 3

- **n4** (integer)

Node number 4

- **n5** (integer)

Node number 5

- **n6** (integer)

Node number 6

- **n7 (optional)** (integer)

Node number 7

- **n8 (optional)** (integer)

Node number 8

Returns

Tshell object

Return type

Tshell

Example

To create a new thick shell in model m with label 100, part 10 and nodes 1, 2, 3, 4, 5, 6, 7, 8:

```
var t = new Tshell(m, 100, 10, 1, 2, 3, 4, 5, 6, 7, 8);
```

Details of functions

AspectRatio()

Description

Calculates the aspect ratio for the thick shell

Arguments

No arguments

Returns

real

Return type

Number

Example

To calculate the aspect ratio for thick shell t:

```
var ratio = t.AspectRatio();
```

AssociateComment(Comment[*Comment*])

Description

Associates a comment with a thick shell.

Arguments

- **Comment** (Comment)

Comment that will be attached to the thick shell

Returns

No return value

Example

To associate comment c to the thick shell t:

```
t.AssociateComment(c);
```

Blank()

Description

Blanks the thick shell

Arguments

No arguments

Returns

No return value

Example

To blank thick shell t:

```
t.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the thick shells in the model.

Arguments

- **Model** (Model)

Model that all thick shells will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the thick shells in model m:

```
Tshell.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged thick shells in the model.

Arguments

- **Model** (Model)

Model that all the flagged thick shells will be blanked in

- **flag** (Flag)

Flag set on the thick shells that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the thick shells in model m flagged with f:

```
Tshell.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the thick shell is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if thick shell t is blanked:

```
if (t.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse thick shell t:

```
t.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the thick shell.

Arguments

- **flag** (Flag)

Flag to clear on the thick shell

Returns

No return value

Example

To clear flag f for thick shell t:

```
t.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the thick shell. The target include of the copied thick shell can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

Tshell object

Return type

Tshell

Example

To copy thick shell t into thick shell z:

```
var z = t.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a thick shell

Arguments

- **Model** (Model)

Model that the thick shell will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

Tshell object (or null if not made).

Return type

Tshell

Example

To start creating a thick shell in model m:

```
var t = Tshell.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a thick shell.

Arguments

- **Comment** (Comment)

Comment that will be detached from the thick shell

Returns

No return value

Example

To detach comment c from the thick shell t:

```
t.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit thick shell t:

```
t.Edit();
```

ElemCut(Database cross section label*[integer]*)

Description

Returns coordinates of the intersections between a thick shell and a database cross section.

Arguments

- **Database cross section label** (integer)

The label of the database cross section.

Returns

Object with the following properties:

Name	Type	Description
face1	Array of reals	An array containing the x1,y1,z1,x2,y2,z2 coordinates of the cut line on the face 1. Null if no cut on this face.

face2	Array of reals	An array containing the x1,y1,z1,x2,y2,z2 coordinates of the cut line on the face 2. Null if no cut on this face.
face3	Array of reals	An array containing the x1,y1,z1,x2,y2,z2 coordinates of the cut line on the face 3. Null if no cut on this face.
face4	Array of reals	An array containing the x1,y1,z1,x2,y2,z2 coordinates of the cut line on the face 4. Null if no cut on this face.
face5	Array of reals	An array containing the x1,y1,z1,x2,y2,z2 coordinates of the cut line on the face 5. Null if no cut on this face.
face6	Array of reals	An array containing the x1,y1,z1,x2,y2,z2 coordinates of the cut line on the face 6. Null if no cut on this face.

Return type

object

Example

To see if the database cross section 200 cuts thick shell s and at which points it cuts face 3 of the thick shell:

```
var data = s.ElemCut(200);
var face = data.face3;
if(face)
{
    var point1_x = face[0];
    var point1_y = face[1];
    var point1_z = face[2];
    var point2_x = face[3];
    var point2_y = face[4];
    var point2_z = face[5];
}
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for thick shell with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The thick shell object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for thick shell t:

```
t.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for thick shell t:

```
t.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for thick shell. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for thick shell t:

```
t.Error("My custom error");
```

ExtractColour()

Description

Extracts the **actual** colour used for thick shell.
By default in PRIMER many entities such as elements get their colour automatically from the part that they are in. PRIMER cycles through 13 default colours based on the label of the entity. In this case the thick shell colour property will return the value Colour.PART instead of the actual colour. This method will return the actual colour which is used for drawing the thick shell.

Arguments

No arguments

Returns

colour value (integer)

Return type

Number

Example

To return the colour used for drawing thick shell t:

```
var colour = t.ExtractColour();
```

FindTshellInBox(Model*[Model]*, xmin*[real]*, xmax*[real]*, ymin*[real]*, ymax*[real]*, zmin*[real]*, zmax*[real]*, flag (optional)*[integer]*, excl (optional)*[integer]*, vis_only (optional)*[integer]*) [static]

Description

Returns an array of Tshell objects for the thick shells within a box. Please note this function provides a list of all thick shells that could potentially be in the box (using computationally cheap bounding box comparison) it is not a rigorous test of whether the thick shellis actually in the box. This may include tshells that are ostensibly outside box. The user should apply their own test. (this function is intended to provide an upper bound of elems to test) Setting the "excl" flag will require that the tshell is fully contained. but this may not capture all the tshells you want to process.

Arguments

- **Model** (Model)

Model designated model

- **xmin** (real)

Minimum bound in global x

- **xmax** (real)

Maximum bound in global x

- **ymin** (real)

Minimum bound in global y

- **ymax** (real)

Maximum bound in global y

- **zmin** (real)

Minimum bound in global z

- **zmax** (real)

Maximum bound in global z

- **flag (optional)** (integer)

Optional flag to restrict thick shells considered, if 0 all tshells considered

- **excl (optional)** (integer)

Optional flag (0) Apply inclusive selection (1) Apply exclusive selection inclusive selection means elements intersect box exclusive selection means elements contained in box

- **vis_only (optional)** (integer)

Optional flag to consider visible elements only (1), if (0) all elements considered

Returns

Array of Tshell objects

Return type

Array

Example

To get an array of Tshell objects for flagged thick shells within defined box (inclusive selection)

```
var s = Tshell.FindTshellInBox(m, xmin, xmax, ymin, ymax, zmin, zmax, flag, 0,
0);
```

First(Model*[Model]*) [static]

Description

Returns the first thick shell in the model.

Arguments

- **Model** (Model)

Model to get first thick shell in

Returns

Tshell object (or null if there are no thick shells in the model).

Return type

Tshell

Example

To get the first thick shell in model m:

```
var t = Tshell.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free thick shell label in the model. Also see Tshell.LastFreeLabel(), Tshell.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free thick shell label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

Tshell label.

Return type

Number

Example

To get the first free thick shell label in model m:

```
var label = Tshell.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the thick shells in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all thick shells will be flagged in

- **flag** (Flag)

Flag to set on the thick shells

Returns

No return value

Example

To flag all of the thick shells with flag f in model m:

```
Tshell.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the thick shell is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the thick shell

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if thick shell t has flag f set on it:

```
if (t.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each thick shell in the model.
Note that ForEach has been designed to make looping over thick shells as fast as possible and so has some limitations.
Firstly, a single temporary Tshell object is created and on each function call it is updated with the current thick shell data. This means that you should not try to store the Tshell object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new thick shells inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all thick shells are in

- **func** (function)

Function to call for each thick shell

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the thick shells in model m:

```
Tshell.ForEach(m, test);
function test(t)
{
// t is Tshell object
}
```

To call function test for all of the thick shells in model m with optional object:

```
var data = { x:0, y:0 };
Tshell.ForEach(m, test, data);
function test(t, extra)
{
// t is Tshell object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Tshell objects or properties for all of the thick shells in a model in PRIMER. If the optional property argument is not given then an array of Tshell objects is returned. If the property argument is given, that property value for each thick shell is returned in the array instead of a Tshell object

Arguments

- **Model** (Model)

Model to get thick shells from

- **property (optional)** (string)

Name for property to get for all thick shells in the model

Returns

Array of Tshell objects or properties

Return type

Array

Example

To make an array of Tshell objects for all of the thick shells in model m:

```
var a = Tshell.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each thick shell in model m:

```
var a = Tshell.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a thick shell.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the thick shell t:

```
var comm_array = t.GetComments();
```

GetCompositeData(ipt*[integer]*)

Description

Returns the composite data for an integration point in *ELEMENT_TSHELL_COMPOSITE.

Arguments

- **ipt** (integer)

The integration point you want the data for. **Note that integration points start at 0, not 1**.

Returns

An array of numbers containing the material id, thickness and beta angle.

Return type

Number

Example

To get the composite data for the 3rd integration point for thick shell t:

```
if (t.composite && s.nip >= 3)
{
    var ipt_data = t.GetCompositeData(2);
}
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of Tshell objects for all of the flagged thick shells in a model in PRIMER If the optional property argument is not given then an array of Tshell objects is returned. If the property argument is given, then that property value for each thick shell is returned in the array instead of a Tshell object

Arguments

- **Model** (Model)

Model to get thick shells from

- **flag** (Flag)

Flag set on the thick shells that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged thick shells in the model

Returns

Array of Tshell objects or properties

Return type

Array

Example

To make an array of Tshell objects for all of the thick shells in model m flagged with f:

```
var t = Tshell.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the thick shells in model m flagged with f:

```
var a = Tshell.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the Tshell object for a thick shell ID.

Arguments

- **Model** (Model)

Model to find the thick shell in

- **number** (integer)

number of the thick shell you want the Tshell object for

Returns

Tshell object (or null if thick shell does not exist).

Return type

Tshell

Example

To get the Tshell object for thick shell 100 in model m

```
var t = Tshell.GetFromID(m, 100);
```

GetNodeIDs()

Description

Returns the labels of the nodes on the thick shell as an array. See also Tshell.GetNodes()

Arguments

No arguments

Returns

Array of node labels (integers)

Return type

Number

Example

To return the node labels of thick shell t as an array

```
var nodes = t.GetNodeIDs();
```

GetNodes()

Description

Returns the nodes on the thick shell as an array of Node objects. See also Tshell.GetNodeIDs()

Arguments

No arguments

Returns

Array of Node objects

Return type

Array

Example

To return the nodes of thick shell t as an array

```
var nodes = t.GetNodes();
```

GetParameter(prop*[string]*)

Description

Checks if a Tshell property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the Tshell.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

thick shell property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if Tshell property t.example is a parameter:

```
Options.property_parameter_names = true;
if (t.GetParameter(t.example) ) do_something...
Options.property_parameter_names = false;
```

To check if Tshell property t.example is a parameter by using the GetParameter method:

```
if (t.ViewParameters().GetParameter(t.example) ) do_something...
```

Jacobian()

Description

Calculates the jacobian for the thick shell

Arguments

No arguments

Returns

real

Return type

Number

Example

To calculate the jacobian for thick shell t:

```
var jacobian = s.Jacobian();
```

Keyword()

Description

Returns the keyword for this thick shell (*ELEMENT_TSHELL or *ELEMENT_TSHELL_COMPOSITE). **Note that a carriage return is not added**. See also Tshell.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for thick shell t:

```
var key = t.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the thick shell. **Note that a carriage return is not added**. See also Tshell.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for thick shell t:

```
var cards = t.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last thick shell in the model.

Arguments

- **Model** (Model)

Model to get last thick shell in

Returns

Tshell object (or null if there are no thick shells in the model).

Return type

Tshell

Example

To get the last thick shell in model m:

```
var t = Tshell.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free thick shell label in the model. Also see Tshell.FirstFreeLabel(), Tshell.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free thick shell label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

Tshell label.

Return type

Number

Example

To get the last free thick shell label in model m:

```
var label = Tshell.LastFreeLabel(m);
```

Next()

Description

Returns the next thick shell in the model.

Arguments

No arguments

Returns

Tshell object (or null if there are no more thick shells in the model).

Return type

Tshell

Example

To get the thick shell in model m after thick shell t:

```
var t = t.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) thick shell label in the model. Also see Tshell.FirstFreeLabel(), Tshell.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free thick shell label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

Tshell label.

Return type

Number

Example

To get the next free thick shell label in model m:

```
var label = Tshell.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a thick shell.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only thick shells from that model can be picked. If the argument is a Flag then only thick shells that are flagged with *limit* can be selected. If omitted, or null, any thick shells from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

Tshell object (or null if not picked)

Return type

Tshell

Example

To pick a thick shell from model m giving the prompt 'Pick thick shell from screen':

```
var t = Tshell.Pick('Pick thick shell from screen', m);
```

Previous()

Description

Returns the previous thick shell in the model.

Arguments

No arguments

Returns

Tshell object (or null if there are no more thick shells in the model).

Return type

Tshell

Example

To get the thick shell in model m before thick shell t:

```
var t = t.Previous();
```

RemoveCompositeData(ipt*[integer]*)

Description

Removes the composite data for an integration point in *ELEMENT_TSHELL_COMPOSITE.

Arguments

- **ipt** (integer)

The integration point you want to remove. **Note that integration points start at 0, not 1**.

Returns

No return value.

Example

To remove the composite data for the 3rd integration point for thick shell t:

```
t.RemoveCompositeData(2);
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the thick shells in the model.

Arguments

- **Model** (Model)

Model that all thick shells will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the thick shells in model m, from 1000000:

```
Tshell.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged thick shells in the model.

Arguments

- **Model** (Model)

Model that all the flagged thick shells will be renumbered in

- **flag** (Flag)

Flag set on the thick shells that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the thick shells in model m flagged with f, from 1000000:

```
Tshell.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select thick shells using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting thick shells

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only thick shells from that model can be selected. If the argument is a Flag then only thick shells that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any thick shells can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of thick shells selected or null if menu cancelled

Return type

Number

Example

To select thick shells from model m, flagging those selected with flag f, giving the prompt 'Select thick shells':

```
Tshell.Select(f, 'Select thick shells', m);
```

To select thick shells, flagging those selected with flag f but limiting selection to thick shells flagged with flag l, giving the prompt 'Select thick shells':

```
Tshell.Select(f, 'Select thick shells', l);
```

SetCompositeData(ipt*[integer]*, mid*[integer]*, thick*[real]*, beta*[real]*)

Description

Sets the composite data for an integration point in *ELEMENT_TSHELL_COMPOSITE.

Arguments

- **ipt** (integer)

The integration point you want to set the data for. **Note that integration points start at 0, not 1**.

- **mid** (integer)

Material ID for the integration point.

- **thick** (real)

Thickness of the integration point.

- **beta** (real)

Material angle of the integration point.

Returns

No return value.

Example

To set the composite data for the 3rd integration point to mat 1, thickness 0.5 and angle 45, for thick shell t:

```
t.SetCompositeData(2, 1, 0.5, 45);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the thick shell.

Arguments

- **flag** (Flag)

Flag to set on the thick shell

Returns

No return value

Example

To set flag f for thick shell t:

```
t.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the thick shell. The thick shell will be sketched until you either call Tshell.Unsketch(), Tshell.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the thick shell is sketched. If omitted redraw is true. If you want to sketch several thick shells and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch thick shell t:

```
t.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged thick shells in the model. The thick shells will be sketched until you either call Tshell.Unsketch(), Tshell.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged thick shells will be sketched in

- **flag** (Flag)

Flag set on the thick shells that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the thick shells are sketched. If omitted redraw is true. If you want to sketch flagged thick shells several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all thick shells flagged with flag in model m:

```
Tshell.SketchFlagged(m, flag);
```

Timestep()

Description

Calculates the timestep for the thick shell

Arguments

No arguments

Returns

real

Return type

Number

Example

To calculate the timestep for thick shell t:

```
var timestep = t.Timestep();
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of thick shells in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing thick shells should be counted. If false or omitted referenced but undefined thick shells will also be included in the total.

Returns

number of thick shells

Return type

Number

Example

To get the total number of thick shells in model m:

```
var total = Tshell.Total(m);
```

Unblank()

Description

Unblanks the thick shell

Arguments

No arguments

Returns

No return value

Example

To unblank thick shell t:

```
t.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the thick shells in the model.

Arguments

- **Model** (Model)

Model that all thick shells will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the thick shells in model m:

```
Tshell.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged thick shells in the model.

Arguments

- **Model** (Model)

Model that the flagged thick shells will be unblanked in

- **flag** (Flag)

Flag set on the thick shells that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the thick shells in model m flagged with f:

```
Tshell.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the thick shells in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all thick shells will be unset in

- **flag** (Flag)

Flag to unset on the thick shells

Returns

No return value

Example

To unset the flag f on all the thick shells in model m:

```
Tshell.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the thick shell.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the thick shell is unsketched. If omitted redraw is true. If you want to unsketch several thick shells and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch thick shell t:

```
t.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all thick shells.

Arguments

- **Model** (Model)

Model that all thick shells will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the thick shells are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all thick shells in model m:

```
Tshell.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged thick shells in the model.

Arguments

- **Model** (Model)

Model that all thick shells will be unsketched in

- **flag** (Flag)

Flag set on the thick shells that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the thick shells are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all thick shells flagged with flag in model m:

```
Tshell.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with ’method chaining’ (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

Tshell object.

Return type

Tshell

Example

To check if Tshell property t.example is a parameter by using the Tshell.GetParameter() method:

```
if (t.ViewParameters().GetParameter(t.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for thick shell with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The thick shell object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for thick shell t:

```
t.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for thick shell t:

```
t.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for thick shell. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for thick shell t:

```
t.Warning("My custom warning");
```

Warpage()

Description

Calculates the warpage for the thick shell

Arguments

No arguments

Returns

real

Return type

Number

Example

To calculate the warpage for thick shell t:

```
var warpage = s.Warpage();
```

Xrefs()

Description

Returns the cross references for this thick shell.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for thick shell t:

```
var xrefs = t.Xrefs();
```

---

toString()

Description

Creates a string containing the thick shell data in keyword format. Note that this contains the keyword header and the keyword cards. See also Tshell.Keyword() and Tshell.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for thick shell t in keyword format

```
var str = t.toString();
```

FreqFRF class

The FreqFRF class gives you access to *FREQUENCY_DOMAIN_FRF keyword in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- First(Model*[Model]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)

Member functions

- AddSubcaseData()
- AssociateComment(Comment*[Comment]*)
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- GetSubcaseData(index*[integer]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- SetSubcaseData(index*[integer]*, title*[string]*, vad1*[integer]*, data*[Array of data]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

FreqFRF constants

Name	Description
FreqFRF.BLANK	FreqFRF option for *FREQUENCY_DOMAIN_FRF_BLANK.
FreqFRF.SUBCASE	FreqFRF option for *FREQUENCY_DOMAIN_FRF_SUBCASE.

Constants for ID type for n1/ n11

Name	Description
FreqFRF.IGA_EDGE_UVW	Property n1typ/ n11typ value EQ.5.0: n1/ n11 is IGA edge UVW ID.
FreqFRF.IGA_EDGE_UVW_SET	Property n1typ/ n11typ value EQ.6.0: n1/ n11 is IGA edge UVW Set ID.
FreqFRF.IGA_FACE_XYZ	Property n1typ/ n11typ value EQ.7.0: n1/ n11 is IGA face XYZ Set ID.
FreqFRF.IGA_FACE_XYZ_SET	Property n1typ/ n11typ value EQ.8.0: n1/ n11 is IGA face XYZ Set ID.

Constants for ID type for n2

Name	Description
FreqFRF.IGA_POINT_UVW	Property n1typ/ n11typ value EQ.3.0: n2 is IGA point UVW ID.
FreqFRF.IGA_POINT_UVW_SET	Property n1typ/ n11typ value EQ.4.0: n2 is IGA point UVW Set ID.
FreqFRF.NODE	Property n1typ/ n2typ value EQ.0.0: n2 is Node ID (default).
FreqFRF.NODE_SET	Property n1typ/ n11typ value EQ.1.0: n2 is Node Set ID.
FreqFRF.SEGMENT_SET	Property n1typ/ n11typ value EQ.2.0: n2 is Segment Set ID.

FreqFRF properties

Name	Type	Description
dampf	real	Modal damping coefficient.
dmpmas	real	Mass proportional damping constant in Rayleigh damping.
dmpstf	real	Stiffness proportional damping constant in Rayleigh damping.
dof1	integer	Applicable degrees-of-freedom for excitation input (ignored if vad1 = 4).
dof2	integer	Applicable degrees-of-freedom for response output.
exists (read only)	logical	true if *FREQUENCY_DOMAIN_FRF exists, false if referred to but not defined.
fmax	real	Maximum frequency for FRF output (cycles/time).
fmin	real	Minimum frequency for FRF output (cycles/time).
fnmax	real	Optional maximum natural frequency employed in FRF computation.
fspace	integer	Frequency spacing option for FRF output.
include	integer	The Include file number that the *FREQUENCY_DOMAIN_FRF is in.
label (read only)	integer	The label the *FREQUENCY_DOMAIN_FRF has in PRIMER.
lcdam	integer	Load Curve ID defining mode dependent modal damping coefficient.
lcfreq	integer	Load Curve ID defining the frequencies for FRF output.
lctyp	integer	Type of load curve defining modal damping coefficient.
mdmax	integer	The last mode employed in FRF computation (optional).
mdmin	integer	The first mode employed in FRF computation (optional).
model (read only)	integer	The Model number that the *FREQUENCY_DOMAIN_FRF is in.
n1	integer	Node/ Node Set/ Segment Set ID for excitation input.

n11	integer	Node/ Node Set/ Segment Set ID for excitation input.
n11typ	integer	Type of n11. Values can be FreqFRF.NODE, FreqFRF.NODE_SET, FreqFRF.SEGMENT_SET, FreqFRF.IGA_POINT_UVW, FreqFRF.IGA_POINT_UVW_SET, FreqFRF.IGA_EDGE_UVW, FreqFRF.IGA_EDGE_UVW_SET, FreqFRF.IGA_FACE_XYZ or FreqFRF.IGA_FACE_XYZ_SET
n1typ	integer	Type of n1. Values can be FreqFRF.NODE, FreqFRF.NODE_SET, FreqFRF.SEGMENT_SET. FreqFRF.IGA_POINT_UVW, FreqFRF.IGA_POINT_UVW_SET, FreqFRF.IGA_EDGE_UVW, FreqFRF.IGA_EDGE_UVW_SET, FreqFRF.IGA_FACE_XYZ or FreqFRF.IGA_FACE_XYZ_SET
n2	integer	Node/ Node Set/ Segment Set ID for response output.
n2typ	integer	Type of n2. Values can be FreqFRF.NODE, FreqFRF.NODE_SET , FreqFRF.SEGMENT_SET, FreqFRF.IGA_POINT_UVW or FreqFRF.IGA_POINT_UVW_SET,
ncases	integer	Number of subcases.
nfreq	integer	Number of frequencies for FRF output.
option	constant	The *FREQUENCY_DOMAIN_FRF option. Can be FreqFRF.BLANK or FreqFRF.SUBCASE
output	integer	Output option.
relatv	integer	Flag for displacement, velocity and acceleration results.
restrt	integer	Restart option.
vad1	integer	Excitation input type.
vad2	integer	Response output type.
vid1	integer	Vector ID (for dof1 = 0) for excitation input.
vid2	integer	Vector ID (for dof2 = 0) for response direction.

Detailed Description

The FreqFRF class allows you to create, modify, edit and manipulate *FREQUENCY_DOMAIN_FRF. See the documentation below for more details.

Constructor

new FreqFRF(Model*[Model]*, option*[constant]*)

Description

Create a new FreqFRF object.

Arguments

- **Model** (Model)

Model that *FREQUENCY_DOMAIN_FRF will be created in.

- **option** (constant)

Specify the type of *FREQUENCY_DOMAIN_FRF. Can be FreqFRF.BLANK, FreqFRF.SUBCASE.

Returns

No return value

Example

To create a new *FREQUENCY_DOMAIN_FRF in model m for BLANK option

```
var f = new FreqFRF(m, FreqFRF.BLANK);
```

new FreqFRF(Model*[Model]*, n1*[integer]*, n1typ*[integer]*, n2*[integer]*, n2typ*[integer]*) [deprecated]

This function is deprecated in version 22.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Create a new FreqFRF object.

Arguments

- **Model** (Model)

Model that *FREQUENCY_DOMAIN_FRF will be created in.

- **n1** (integer)

Node/ Node Set/ Segment Set ID for excitation input.

- **n1typ** (integer)

Type of n1. Values can be FreqFRF.NODE, FreqFRF.NODE_SET or FreqFRF.SEGMENT_SET.

- **n2** (integer)

Node/ Node Set/ Segment Set ID for response output.

- **n2typ** (integer)

Type of n2. Values can be FreqFRF.NODE, FreqFRF.NODE_SET or FreqFRF.SEGMENT_SET.

Returns

FreqFRF object

Return type

FreqFRF

Example

To create a new *FREQUENCY_DOMAIN_FRF in model m with node 10 for excitation output and node set 2 for response output

```
var f = new FreqFRF(m, 10, FreqFRF.NODE, 2, FreqFRF.NODE_SET);
```

Details of functions

AddSubcaseData()

Description

Allows user to add new subcase cards in *FREQUENCY_DOMAIN_SSFRF. This method is only applicable when option is FreqFRF.SUBCASE.

The new cards have uninitialised fields and should be updated by FreqFRF.SetSubcaseData().

Arguments

No arguments

Returns

Index of the new subcase

Return type

integer

Example

To add a new load data card in *FREQUENCY_DOMAIN_FRF f:

```
f.AddSubcaseData();
```

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a *FREQUENCY_DOMAIN_FRF.

Arguments

- **Comment** (Comment)

Comment that will be attached to the *FREQUENCY_DOMAIN_FRF

Returns

No return value

Example

To associate comment c to the *FREQUENCY_DOMAIN_FRF f:

```
f.AssociateComment(c);
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse *FREQUENCY_DOMAIN_FRF f:

```
f.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the *FREQUENCY_DOMAIN_FRF.

Arguments

- **flag** (Flag)

Flag to clear on the *FREQUENCY_DOMAIN_FRF

Returns

No return value

Example

To clear flag f for *FREQUENCY_DOMAIN_FRF f:

```
f.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the *FREQUENCY_DOMAIN_FRF. The target include of the copied *FREQUENCY_DOMAIN_FRF can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

FreqFRF object

Return type

FreqFRF

Example

To copy *FREQUENCY_DOMAIN_FRF f into *FREQUENCY_DOMAIN_FRF z:

```
var z = f.Copy();
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a *FREQUENCY_DOMAIN_FRF.

Arguments

- **Comment** (Comment)

Comment that will be detached from the *FREQUENCY_DOMAIN_FRF

Returns

No return value

Example

To detach comment c from the *FREQUENCY_DOMAIN_FRF f:

```
f.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit *FREQUENCY_DOMAIN_FRF f:

```
f.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for *FREQUENCY_DOMAIN_FRF with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The *FREQUENCY_DOMAIN_FRF object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for *FREQUENCY_DOMAIN_FRF f:

```
f.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for *FREQUENCY_DOMAIN_FRF f:

```
f.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for *FREQUENCY_DOMAIN_FRF. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for *FREQUENCY_DOMAIN_FRF f:

```
f.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first *FREQUENCY_DOMAIN_FRF in the model.

Arguments

- **Model** (Model)

Model to get first *FREQUENCY_DOMAIN_FRF in

Returns

FreqFRF object (or null if there are no *FREQUENCY_DOMAIN_FRFs in the model).

Return type

FreqFRF

Example

To get the first *FREQUENCY_DOMAIN_FRF in model m:

```
var f = FreqFRF.First(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the *FREQUENCY_DOMAIN_FRFs in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all *FREQUENCY_DOMAIN_FRFs will be flagged in

- **flag** (Flag)

Flag to set on the *FREQUENCY_DOMAIN_FRFs

Returns

No return value

Example

To flag all of the *FREQUENCY_DOMAIN_FRFs with flag f in model m:

```
FreqFRF.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the *FREQUENCY_DOMAIN_FRF is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the *FREQUENCY_DOMAIN_FRF

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if *FREQUENCY_DOMAIN_FRF f has flag f set on it:

```
if (f.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each *FREQUENCY_DOMAIN_FRF in the model.
Note that ForEach has been designed to make looping over *FREQUENCY_DOMAIN_FRFs as fast as possible and so has some limitations.
Firstly, a single temporary FreqFRF object is created and on each function call it is updated with the current *FREQUENCY_DOMAIN_FRF data. This means that you should not try to store the FreqFRF object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new *FREQUENCY_DOMAIN_FRFs inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all *FREQUENCY_DOMAIN_FRFs are in

- **func** (function)

Function to call for each *FREQUENCY_DOMAIN_FRF

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the *FREQUENCY_DOMAIN_FRFs in model m:

```
FreqFRF.ForEach(m, test);
function test(f)
{
// f is FreqFRF object
}
```

To call function test for all of the *FREQUENCY_DOMAIN_FRFs in model m with optional object:

```
var data = { x:0, y:0 };
FreqFRF.ForEach(m, test, data);
function test(f, extra)
{
// f is FreqFRF object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of FreqFRF objects or properties for all of the *FREQUENCY_DOMAIN_FRFs in a model in PRIMER. If the optional property argument is not given then an array of FreqFRF objects is returned. If the property argument is given, that property value for each *FREQUENCY_DOMAIN_FRF is returned in the array instead of a FreqFRF object

Arguments

- **Model** (Model)

Model to get *FREQUENCY_DOMAIN_FRFs from

- **property (optional)** (string)

Name for property to get for all *FREQUENCY_DOMAIN_FRFs in the model

Returns

Array of FreqFRF objects or properties

Return type

Array

Example

To make an array of FreqFRF objects for all of the *FREQUENCY_DOMAIN_FRFs in model m:

```
var a = FreqFRF.GetAll(m);
```

To return an array containing the value of property ’foo’ (for example ’x’ for a node) for each *FREQUENCY_DOMAIN_FRF in model m:

```
var a = FreqFRF.GetAll(m, ’foo’);
```

GetComments()

Description

Extracts the comments associated to a *FREQUENCY_DOMAIN_FRF.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the *FREQUENCY_DOMAIN_FRF f:

```
var comm_array = f.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of FreqFRF objects for all of the flagged *FREQUENCY_DOMAIN_FRFs in a model in PRIMER If the optional property argument is not given then an array of FreqFRF objects is returned. If the property argument is given, then that property value for each *FREQUENCY_DOMAIN_FRF is returned in the array instead of a FreqFRF object

Arguments

- **Model** (Model)

Model to get *FREQUENCY_DOMAIN_FRFs from

- **flag** (Flag)

Flag set on the *FREQUENCY_DOMAIN_FRFs that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged *FREQUENCY_DOMAIN_FRFs in the model

Returns

Array of FreqFRF objects or properties

Return type

Array

Example

To make an array of FreqFRF objects for all of the *FREQUENCY_DOMAIN_FRFs in model m flagged with f:

```
var f = FreqFRF.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the *FREQUENCY_DOMAIN_FRFs in model m flagged with f:

```
var a = FreqFRF.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the FreqFRF object for a *FREQUENCY_DOMAIN_FRF ID.

Arguments

- **Model** (Model)

Model to find the *FREQUENCY_DOMAIN_FRF in

- **number** (integer)

number of the *FREQUENCY_DOMAIN_FRF you want the FreqFRF object for

Returns

FreqFRF object (or null if *FREQUENCY_DOMAIN_FRF does not exist).

Return type

FreqFRF

Example

To get the FreqFRF object for *FREQUENCY_DOMAIN_FRF 100 in model m

```
var f = FreqFRF.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a FreqFRF property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the FreqFRF.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

*FREQUENCY_DOMAIN_FRF property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if FreqFRF property f.example is a parameter:

```
Options.property_parameter_names = true;
if (f.GetParameter(f.example) ) do_something...
Options.property_parameter_names = false;
```

To check if FreqFRF property f.example is a parameter by using the GetParameter method:

```
if (f.ViewParameters().GetParameter(f.example) ) do_something...
```

GetSubcaseData(index*[integer]*)

Description

Returns the data for a specific subcase as an array. For each subcase there will be 13 values when vad1=12 else 11 values . There can be as many subcases as needed.

This method is only applicable when option is FreqFRF.SUBCASE.

Arguments

- **index** (integer)

Index you want the subcase data for. **Note that indices start at 0**.

Returns

An array containing the subcase data (values: title[string], n1[integer], n1typ[integer], n1typ[integer], dof1[integer], vad1[integer], vid1[integer], n2[integer], n2typ[integer],dof2[integer], vad2[integer], vid2[integer], n11[integer], n11typ[integer])

n11 and n11typ are present only when vad1 =12

Return type

Number

Example

To get the data for the 2nd subcase for *FREQUENCY_DOMAIN_FRF f:

```
var sdata = f.GetSubcaseData(1);
```

Keyword()

Description

Returns the keyword for this *FREQUENCY_DOMAIN_FRF **Note that a carriage return is not added**. See also FreqFRF.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for *FREQUENCY_DOMAIN_FRF f:

```
var key = f.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the *FREQUENCY_DOMAIN_FRF. **Note that a carriage return is not added**. See also FreqFRF.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for *FREQUENCY_DOMAIN_FRF f:

```
var cards = f.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last *FREQUENCY_DOMAIN_FRF in the model.

Arguments

- **Model** (Model)

Model to get last *FREQUENCY_DOMAIN_FRF in

Returns

FreqFRF object (or null if there are no *FREQUENCY_DOMAIN_FRFs in the model).

Return type

FreqFRF

Example

To get the last *FREQUENCY_DOMAIN_FRF in model m:

```
var f = FreqFRF.Last(m);
```

Next()

Description

Returns the next *FREQUENCY_DOMAIN_FRF in the model.

Arguments

No arguments

Returns

FreqFRF object (or null if there are no more *FREQUENCY_DOMAIN_FRFs in the model).

Return type

FreqFRF

Example

To get the *FREQUENCY_DOMAIN_FRF in model m after *FREQUENCY_DOMAIN_FRF f:

```
var f = f.Next();
```

Previous()

Description

Returns the previous *FREQUENCY_DOMAIN_FRF in the model.

Arguments

No arguments

Returns

FreqFRF object (or null if there are no more *FREQUENCY_DOMAIN_FRFs in the model).

Return type

FreqFRF

Example

To get the *FREQUENCY_DOMAIN_FRF in model m before *FREQUENCY_DOMAIN_FRF f:

```
var f = f.Previous();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select *FREQUENCY_DOMAIN_FRFs using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting *FREQUENCY_DOMAIN_FRFs

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only *FREQUENCY_DOMAIN_FRFs from that model can be selected. If the argument is a Flag then only *FREQUENCY_DOMAIN_FRFs that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any *FREQUENCY_DOMAIN_FRFs can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of *FREQUENCY_DOMAIN_FRFs selected or null if menu cancelled

Return type

Number

Example

To select *FREQUENCY_DOMAIN_FRFs from model m, flagging those selected with flag f, giving the prompt 'Select *FREQUENCY_DOMAIN_FRFs':

```
FreqFRF.Select(f, 'Select *FREQUENCY_DOMAIN_FRFs', m);
```

To select *FREQUENCY_DOMAIN_FRFs, flagging those selected with flag f but limiting selection to *FREQUENCY_DOMAIN_FRFs flagged with flag l, giving the prompt 'Select *FREQUENCY_DOMAIN_FRFs':

```
FreqFRF.Select(f, 'Select *FREQUENCY_DOMAIN_FRFs', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the *FREQUENCY_DOMAIN_FRF.

Arguments

- **flag** (Flag)

Flag to set on the *FREQUENCY_DOMAIN_FRF

Returns

No return value

Example

To set flag f for *FREQUENCY_DOMAIN_FRF f:

```
f.SetFlag(f);
```

SetSubcaseData(index*[integer]*, title*[string]*, vad1*[integer]*, data*[Array of data]*)

Description

Set the data for a specific subcase. There can be as many subcases as needed.

This method is only applicable when option is FreqFRF.SUBCASE.

Arguments

- **index** (integer)

Index you want to set subcase data for. **Note that indices start at 0**.

- **title** (string)

A description of the current subcase (can be blank).

- **vad1** (integer)

Value of vad1.

- **data** (Array of data)

An array containing the subcase data (values: n1[integer], n1typ[integer], dof1[integer], vad1[integer], vid1[integer], n2[integer], n2typ[integer], dof2[integer], vad2[integer], vid2[integer])

For vad1=12 Extra 2 arguments to be given n1[integer], n11typ[integer]

Returns

No return value.

Example

To set the 2nd subcase data for *FREQUENCY_DOMAIN_FRF to have title "subcase 1" and vad1 as 2:

```
f.SetSubcaseData(1, "subcase 1", 2, ldata);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of *FREQUENCY_DOMAIN_FRFs in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing *FREQUENCY_DOMAIN_FRFs should be counted. If false or omitted referenced but undefined *FREQUENCY_DOMAIN_FRFs will also be included in the total.

Returns

number of *FREQUENCY_DOMAIN_FRFs

Return type

Number

Example

To get the total number of *FREQUENCY_DOMAIN_FRFs in model m:

```
var total = FreqFRF.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the *FREQUENCY_DOMAIN_FRFs in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all *FREQUENCY_DOMAIN_FRFs will be unset in

- **flag** (Flag)

Flag to unset on the *FREQUENCY_DOMAIN_FRFs

Returns

No return value

Example

To unset the flag f on all the *FREQUENCY_DOMAIN_FRFs in model m:

```
FreqFRF.UnflagAll(m, f);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

FreqFRF object.

Return type

FreqFRF

Example

To check if FreqFRF property f.example is a parameter by using the FreqFRF.GetParameter() method:

```
if (f.ViewParameters().GetParameter(f.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for *FREQUENCY_DOMAIN_FRF with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The *FREQUENCY_DOMAIN_FRF object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for *FREQUENCY_DOMAIN_FRF f:

```
f.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for *FREQUENCY_DOMAIN_FRF f:

```
f.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for *FREQUENCY_DOMAIN_FRF. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for *FREQUENCY_DOMAIN_FRF f:

```
f.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this *FREQUENCY_DOMAIN_FRF.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for *FREQUENCY_DOMAIN_FRF f:

```
var xrefs = f.Xrefs();
```

---

toString()

Description

Creates a string containing the *FREQUENCY_DOMAIN_FRF data in keyword format. Note that this contains the keyword header and the keyword cards. See also FreqFRF.Keyword() and FreqFRF.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for *FREQUENCY_DOMAIN_FRF f in keyword format

```
var str = f.toString();
```

FreqSSD class

The FreqSSD class gives you access to *FREQUENCY_DOMAIN_SSD keyword in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)

Member functions

- AddLoadData()
- AddSubcaseData()
- AssociateComment(Comment*[Comment]*)
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetErpData(index*[integer]*)
- GetLoadData(index*[integer]*)
- GetParameter(prop*[string]*)
- GetSubcaseData(index*[integer]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- RemoveLoadData(index*[integer]*)
- RemoveSubcaseData(index*[integer]*)
- SetErpData(index*[integer]*, data*[Array of data]*)
- SetFlag(flag*[Flag]*)
- SetLoadData(index*[integer]*, data*[Array of data]*)
- SetSubcaseData(index*[integer]*, caseid*[string]*, title*[string]*, nload*[integer]*, data*[Array of data]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

FreqSSD constants

Name	Description

FreqSSD.DIRECT	FreqSSD option for *FREQUENCY_DOMAIN_SSD_DIRECT.
FreqSSD.DIRECT_FD	FreqSSD option for *FREQUENCY_DOMAIN_SSD_DIRECT_FREQUENCY_DEPENDENT.
FreqSSD.ERP	FreqSSD option for *FREQUENCY_DOMAIN_SSD_ERP.
FreqSSD.FATIGUE	FreqSSD option for *FREQUENCY_DOMAIN_SSD_FATIGUE.
FreqSSD.FRF	FreqSSD option for *FREQUENCY_DOMAIN_SSD_FRF.
FreqSSD.SUBCASE	FreqSSD option for *FREQUENCY_DOMAIN_SSD_SUBCASE.

FreqSSD properties

Name	Type	Description
c	real	Sound speed of the fluid (for option FreqSSD.ERP only).
dampf	real	Modal damping coefficient.
dmpflg	integer	Damping flag.
dmpmas	real	Mass proportional damping constant in Rayleigh damping.
dmpstf	real	Stiffness proportional damping constant in Rayleigh damping.
erpref	real	ERP reference value (for option FreqSSD.ERP only).
erprlf	real	ERP radiation loss factor (for option FreqSSD.ERP only).
exists (read only)	logical	true if *FREQUENCY_DOMAIN_SSD exists, false if referred to but not defined.
fnmax	real	The maximum natural frequency in modal superposition method (optional).
fnmin	real	The minimum natural frequency in modal superposition method (optional).
include	integer	The Include file number that the *FREQUENCY_DOMAIN_SSD is in.
istress	integer	Stress computation flag (for option FreqSSD.DIRECT only).
label (read only)	integer	The label the *FREQUENCY_DOMAIN_SSD has in PRIMER.
lcdam	integer	Load Curve ID defining mode dependent modal damping coefficient.
lcflag	integer	Load curve definition flag.
lcftg	integer	Load Curve ID defining duration of excitation for each frequency (for option FreqSSD.FATIGUE only).
lctyp	integer	Type of load curve defining modal damping coefficient.
mdmax	integer	The last mode in modal superposition method (optional).
mdmin	integer	The first mode in modal superposition method (optional).
memory	integer	Memory flag.
model (read only)	integer	The Model number that the *FREQUENCY_DOMAIN_SSD is in.
nerp	integer	Number of ERP panels.
notyp	integer	Type of NOUT.
nout	integer	Part, part set, segment set, or node set ID for response output.
nova	integer	Response output type.

option	constant	The *FREQUENCY_DOMAIN_SSD option. Can be FreqSSD.DIRECT, FreqSSD.DIRECT_FD, FreqSSD.FATIGUE, FreqSSD.FRF, FreqSSD.ERP or FreqSSD.SUBCASE.
radeff	integer	Radiation efficiency computation flag (for option FreqSSD.ERP only).
relatv	integer	Flag for displacement, velocity and acceleration results.
restdp	integer	Restart option.
restmd	integer	Restart option.
ro	real	Fluid density (for option FreqSSD.ERP only).
strtyp	integer	Stress used in fatigue analysis.

Detailed Description

The FreqSSD class allows you to create, modify, edit and manipulate *FREQUENCY_DOMAIN_SSD. See the documentation below for more details.

Constructor

new FreqSSD(Model*[Model]*, option*[constant]*)

Description

Create a new FreqSSD object.

Arguments

- **Model** (Model)

Model that *FREQUENCY_DOMAIN_SSD will be created in.

- **option** (constant)

Specify the type of *FREQUENCY_DOMAIN_SSD. Can be FreqSSD.DIRECT, FreqSSD.DIRECT_FD, FreqSSD.FATIGUE, FreqSSD.FRF, FreqSSD.ERP or FreqSSD.SUBCASE.

Returns

FreqSSD object

Return type

FreqSSD

Example

To create a new *FREQUENCY_DOMAIN_SSD in model m, of type FATIGUE

```
var f = new FreqSSD(m, FreqSSD.FATIGUE);
```

Details of functions

AddLoadData()

Description

Allows user to add a new load card in *FREQUENCY_DOMAIN_SSD. This method is only applicable when option is not FreqSSD.SUBCASE.

The new card has uninitialised fields and should be updated by FreqSSD.SetLoadData().

Arguments

No arguments

Returns

Index of the new load

Return type

integer

Example

To add a new load data card in *FREQUENCY_DOMAIN_SSD f:

```
f.AddLoadData();
```

AddSubcaseData()

Description

Allows user to add new subcase cards in *FREQUENCY_DOMAIN_SSD. This method is only applicable when option is FreqSSD.SUBCASE.

The new cards have uninitialised fields and should be updated by FreqSSD.SetSubcaseData().

Arguments

No arguments

Returns

Index of the new subcase

Return type

integer

Example

To add a new load data card in *FREQUENCY_DOMAIN_SSD f:

```
f.AddSubcaseData();
```

AssociateComment(Comment[*Comment*])

Description

Associates a comment with a *FREQUENCY_DOMAIN_SSD.

Arguments

- **Comment** (Comment)

Comment that will be attached to the *FREQUENCY_DOMAIN_SSD

Returns

No return value

Example

To associate comment c to the *FREQUENCY_DOMAIN_SSD f:

```
f.AssociateComment(c);
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse *FREQUENCY_DOMAIN_SSD f:

```
f.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the *FREQUENCY_DOMAIN_SSD.

Arguments

- **flag** (Flag)

Flag to clear on the *FREQUENCY_DOMAIN_SSD

Returns

No return value

Example

To clear flag f for *FREQUENCY_DOMAIN_SSD f:

```
f.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the *FREQUENCY_DOMAIN_SSD. The target include of the copied *FREQUENCY_DOMAIN_SSD can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

FreqSSD object

Return type

FreqSSD

Example

To copy *FREQUENCY_DOMAIN_SSD f into *FREQUENCY_DOMAIN_SSD z:

```
var z = f.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a *FREQUENCY_DOMAIN_SSD

Arguments

- **Model** (Model)

Model that the *FREQUENCY_DOMAIN_SSD will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

FreqSSD object (or null if not made).

Return type

FreqSSD

Example

To start creating a *FREQUENCY_DOMAIN_SSD in model m:

```
var f = FreqSSD.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a *FREQUENCY_DOMAIN_SSD.

Arguments

- **Comment** (Comment)

Comment that will be detached from the *FREQUENCY_DOMAIN_SSD

Returns

No return value

Example

To detach comment c from the *FREQUENCY_DOMAIN_SSD f:

```
f.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit *FREQUENCY_DOMAIN_SSD f:

```
f.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for *FREQUENCY_DOMAIN_SSD with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The *FREQUENCY_DOMAIN_SSD object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for *FREQUENCY_DOMAIN_SSD f:

```
f.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for *FREQUENCY_DOMAIN_SSD f:

```
f.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for *FREQUENCY_DOMAIN_SSD. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for *FREQUENCY_DOMAIN_SSD f:

```
f.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first *FREQUENCY_DOMAIN_SSD in the model.

Arguments

- **Model** (Model)

Model to get first *FREQUENCY_DOMAIN_SSD in

Returns

FreqSSD object (or null if there are no *FREQUENCY_DOMAIN_SSDs in the model).

Return type

FreqSSD

Example

To get the first *FREQUENCY_DOMAIN_SSD in model m:

```
var f = FreqSSD.First(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the *FREQUENCY_DOMAIN_SSDs in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all *FREQUENCY_DOMAIN_SSDs will be flagged in

- **flag** (Flag)

Flag to set on the *FREQUENCY_DOMAIN_SSDs

Returns

No return value

Example

To flag all of the *FREQUENCY_DOMAIN_SSDs with flag f in model m:

```
FreqSSD.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the *FREQUENCY_DOMAIN_SSD is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the *FREQUENCY_DOMAIN_SSD

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if *FREQUENCY_DOMAIN_SSD f has flag f set on it:

```
if (f.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each *FREQUENCY_DOMAIN_SSD in the model.
Note that ForEach has been designed to make looping over *FREQUENCY_DOMAIN_SSDs as fast as possible and so has some limitations.
Firstly, a single temporary FreqSSD object is created and on each function call it is updated with the current *FREQUENCY_DOMAIN_SSD data. This means that you should not try to store the FreqSSD object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new *FREQUENCY_DOMAIN_SSDs inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all *FREQUENCY_DOMAIN_SSDs are in

- **func** (function)

Function to call for each *FREQUENCY_DOMAIN_SSD

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the *FREQUENCY_DOMAIN_SSDs in model m:

```
FreqSSD.ForEach(m, test);
function test(f)
{
// f is FreqSSD object
}
```

To call function test for all of the *FREQUENCY_DOMAIN_SSDs in model m with optional object:

```
var data = { x:0, y:0 };
FreqSSD.ForEach(m, test, data);
function test(f, extra)
{
// f is FreqSSD object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of FreqSSD objects or properties for all of the *FREQUENCY_DOMAIN_SSDs in a model in PRIMER. If the optional property argument is not given then an array of FreqSSD objects is returned. If the property argument is given, that property value for each *FREQUENCY_DOMAIN_SSD is returned in the array instead of a FreqSSD object

Arguments

- **Model** (Model)

Model to get *FREQUENCY_DOMAIN_SSDs from

- **property (optional)** (string)

Name for property to get for all *FREQUENCY_DOMAIN_SSDs in the model

Returns

Array of FreqSSD objects or properties

Return type

Array

Example

To make an array of FreqSSD objects for all of the *FREQUENCY_DOMAIN_SSDs in model m:

```
var a = FreqSSD.GetAll(m);
```

To return an array containing the value of property ’foo’ (for example ’x’ for a node) for each *FREQUENCY_DOMAIN_SSD in model m:

```
var a = FreqSSD.GetAll(m, ’foo’);
```

GetComments()

Description

Extracts the comments associated to a *FREQUENCY_DOMAIN_SSD.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the *FREQUENCY_DOMAIN_SSD f:

```
var comm_array = f.GetComments();
```

GetErpData(index*[integer]*)

Description

Returns the ERP data for a specific ERP part as an array. For each ERP part there will be 2 values. There are nerp ERP parts.

This method is only applicable when option is FreqSSD.ERP.

Arguments

- **index** (integer)

Index you want the ERP data for. **Note that indices start at 0**.

Returns

An array containing the ERP data (values: pid[integer], ptyp[integer]). The array length will be 2.

Return type

Number

Example

To get the data for the 3rd ERP part for *FREQUENCY_DOMAIN_SSD f:

```
var edata = f.GetErpData(2);
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of FreqSSD objects for all of the flagged *FREQUENCY_DOMAIN_SSDs in a model in PRIMER If the optional property argument is not given then an array of FreqSSD objects is returned. If the property argument is given, then that property value for each *FREQUENCY_DOMAIN_SSD is returned in the array instead of a FreqSSD object

Arguments

- **Model** (Model)

Model to get *FREQUENCY_DOMAIN_SSDs from

- **flag** (Flag)

Flag set on the *FREQUENCY_DOMAIN_SSDs that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged *FREQUENCY_DOMAIN_SSDs in the model

Returns

Array of FreqSSD objects or properties

Return type

Array

Example

To make an array of FreqSSD objects for all of the *FREQUENCY_DOMAIN_SSDs in model m flagged with f:

```
var f = FreqSSD.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the *FREQUENCY_DOMAIN_SSDs in model m flagged with f:

```
var a = FreqSSD.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the FreqSSD object for a *FREQUENCY_DOMAIN_SSD ID.

Arguments

- **Model** (Model)

Model to find the *FREQUENCY_DOMAIN_SSD in

- **number** (integer)

number of the *FREQUENCY_DOMAIN_SSD you want the FreqSSD object for

Returns

FreqSSD object (or null if *FREQUENCY_DOMAIN_SSD does not exist).

Return type

FreqSSD

Example

To get the FreqSSD object for *FREQUENCY_DOMAIN_SSD 100 in model m

```
var f = FreqSSD.GetFromID(m, 100);
```

GetLoadData(index*[integer]*)

Description

Returns the data for a specific excitation load as an array. For each load there will be 8 values. There can be as many loads as needed.

This method is only applicable when option is not FreqSSD.SUBCASE.

Arguments

- **index** (integer)

Index you want the load data for. **Note that indices start at 0**.

Returns

An array containing the load data (values: nid[integer], ntyp[integer], dof[integer], vad[integer], lc1[integer], lc2[integer], sf[real], vid[integer]).

The array length will be 8.

Return type

Number

Example

To get the data for the 4th load for *FREQUENCY_DOMAIN_SSD f:

```
var ldata = f.GetLoadData(3);
```

GetParameter(prop*[string]*)

Description

Checks if a FreqSSD property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the FreqSSD.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

*FREQUENCY_DOMAIN_SSD property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if FreqSSD property f.example is a parameter:

```
Options.property_parameter_names = true;
if (f.GetParameter(f.example) ) do_something...
Options.property_parameter_names = false;
```

To check if FreqSSD property f.example is a parameter by using the GetParameter method:

```
if (f.ViewParameters().GetParameter(f.example) ) do_something...
```

GetSubcaseData(index*[integer]*)

Description

Returns the data for a specific subcase as an array. For each subcase there will be 3 + 8 x nload values. There can be as many subcases as needed.

This method is only applicable when option is FreqSSD.SUBCASE.

Arguments

- **index** (integer)

Index you want the subcase data for. **Note that indices start at 0**.

Returns

An array containing the subcase data (values: caseid[string], title[string], nload[integer], nid[integer], ntyp[integer], dof[integer], vad[integer], lc1[integer], lc2[integer], sf[real], vid[integer], ...)

Where values nid to vid are repeated nload times in the array. The array length will be 3 + 8 x nload.

Return type

Number

Example

To get the data for the 2nd subcase for *FREQUENCY_DOMAIN_SSD f:

```
var sdata = f.GetSubcaseData(1);
```

Keyword()

Description

Returns the keyword for this *FREQUENCY_DOMAIN_SSD. **Note that a carriage return is not added**. See also FreqSSD.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for *FREQUENCY_DOMAIN_SSD f:

```
var key = f.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the *FREQUENCY_DOMAIN_SSD. **Note that a carriage return is not added**. See also FreqSSD.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for *FREQUENCY_DOMAIN_SSD f:

```
var cards = f.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last *FREQUENCY_DOMAIN_SSD in the model.

Arguments

- **Model** (Model)

Model to get last *FREQUENCY_DOMAIN_SSD in

Returns

FreqSSD object (or null if there are no *FREQUENCY_DOMAIN_SSDs in the model).

Return type

FreqSSD

Example

To get the last *FREQUENCY_DOMAIN_SSD in model m:

```
var f = FreqSSD.Last(m);
```

Next()

Description

Returns the next *FREQUENCY_DOMAIN_SSD in the model.

Arguments

No arguments

Returns

FreqSSD object (or null if there are no more *FREQUENCY_DOMAIN_SSDs in the model).

Return type

FreqSSD

Example

To get the *FREQUENCY_DOMAIN_SSD in model m after *FREQUENCY_DOMAIN_SSD f:

```
var f = f.Next();
```

Previous()

Description

Returns the previous *FREQUENCY_DOMAIN_SSD in the model.

Arguments

No arguments

Returns

FreqSSD object (or null if there are no more *FREQUENCY_DOMAIN_SSDs in the model).

Return type

FreqSSD

Example

To get the *FREQUENCY_DOMAIN_SSD in model m before *FREQUENCY_DOMAIN_SSD f:

```
var f = f.Previous();
```

RemoveLoadData(index*[integer]*)

Description

Allows user to remove a specified load card in *FREQUENCY_DOMAIN_SSD.

This method is only applicable when option is not FreqSSD.SUBCASE.

Arguments

- **index** (integer)

Index of the load card you want to remove. **Note that indices start at 0**.

Returns

No return value

Example

To remove first load card in *FREQUENCY_DOMAIN_SSD f:

```
f.RemoveLoadData(0);
```

RemoveSubcaseData(index*[integer]*)

Description

Allows user to remove cards for a specified subcase in *FREQUENCY_DOMAIN_SSD.

This method is only applicable when option is FreqSSD.SUBCASE.

Arguments

- **index** (integer)

Index of the subcase you want to remove cards for. **Note that indices start at 0**.

Returns

No return value

Example

To remove cards corresponding to the second subcase in *FREQUENCY_DOMAIN_SSD f:

```
f.RemoveSubcaseData(1);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select *FREQUENCY_DOMAIN_SSDs using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting *FREQUENCY_DOMAIN_SSDs

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only *FREQUENCY_DOMAIN_SSDs from that model can be selected. If the argument is a Flag then only *FREQUENCY_DOMAIN_SSDs that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any *FREQUENCY_DOMAIN_SSDs can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of *FREQUENCY_DOMAIN_SSDs selected or null if menu cancelled

Return type

Number

Example

To select *FREQUENCY_DOMAIN_SSDs from model m, flagging those selected with flag f, giving the prompt 'Select *FREQUENCY_DOMAIN_SSDs':

```
FreqSSD.Select(f, 'Select *FREQUENCY_DOMAIN_SSDs', m);
```

To select *FREQUENCY_DOMAIN_SSDs, flagging those selected with flag f but limiting selection to *FREQUENCY_DOMAIN_SSDs flagged with flag l, giving the prompt 'Select *FREQUENCY_DOMAIN_SSDs':

```
FreqSSD.Select(f, 'Select *FREQUENCY_DOMAIN_SSDs', l);
```

SetErpData(index*[integer]*, data*[Array of data]*)

Description

Set the data for a specific ERP part. For each ERP part there will be 2 values. There are nerp ERP parts.

This method is only applicable when option is FreqSSD.ERP.

Arguments

- **index** (integer)

Index you want to set ERP data for. **Note that indices start at 0**.

- **data** (Array of data)

An array containing the ERP data (values: pid[integer], ptyp[integer]). The array length should be 2.

Returns

No return value.

Example

To set the 3rd ERP part data for *FREQUENCY_DOMAIN_SSD f to the values in array edata:

```
f.SetErpData(2, edata);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the *FREQUENCY_DOMAIN_SSD.

Arguments

- **flag** (Flag)

Flag to set on the *FREQUENCY_DOMAIN_SSD

Returns

No return value

Example

To set flag f for *FREQUENCY_DOMAIN_SSD f:

```
f.SetFlag(f);
```

SetLoadData(index*[integer]*, data*[Array of data]*)

Description

Set the data for a specific excitation load. For each load there will be 8 values. There can be as many loads as needed.

This method is only applicable when option is not FreqSSD.SUBCASE.

Arguments

- **index** (integer)

Index you want to set load data for. **Note that indices start at 0**.

- **data** (Array of data)

An array containing the load data (values: nid[integer], ntyp[integer], dof[integer], vad[integer], lc1[integer], lc2[integer], sf[real], vid[integer]).

The array length should be 8.

Returns

No return value.

Example

To set the 4th load data for *FREQUENCY_DOMAIN_SSD f to the values in array ldata:

```
f.SetLoadData(3, ldata);
```

SetSubcaseData(index*[integer]*, caseid*[string]*, title*[string]*, nload*[integer]*, data*[Array of data]*)

Description

Set the data for a specific subcase. For each subcase, data will have 8 x nload values. There can be as many subcases as needed.

This method is only applicable when option is FreqSSD.SUBCASE.

Arguments

- **index** (integer)

Index you want to set subcase data for. **Note that indices start at 0**.

- **caseid** (string)

Identification string to be used as the case ID (must include at least one letter).

- **title** (string)

A description of the current loading case (can be blank).

- **nload** (integer)

Number of loads for this loading case.

- **data** (Array of data)

An array containing the subcase load data (values: nid[integer], ntyp[integer], dof[integer], vad[integer], lc1[integer], lc2[integer], sf[real], vid[integer], ...)

Where values nid to vid are repeated nload times in the array. The array length should be 8 x nload.

Returns

No return value.

Example

To set the 2nd subcase data for *FREQUENCY_DOMAIN_SSD f to have caseid "ID2", no title, 2 load cards and load data of ldata:

```
f.SetSubcaseData(1, "ID2", "", 2, ldata);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of *FREQUENCY_DOMAIN_SSDs in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing *FREQUENCY_DOMAIN_SSDs should be counted. If false or omitted referenced but undefined *FREQUENCY_DOMAIN_SSDs will also be included in the total.

Returns

number of *FREQUENCY_DOMAIN_SSDs

Return type

Number

Example

To get the total number of *FREQUENCY_DOMAIN_SSDs in model m:

```
var total = FreqSSD.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the *FREQUENCY_DOMAIN_SSDs in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all *FREQUENCY_DOMAIN_SSDs will be unset in

- **flag** (Flag)

Flag to unset on the *FREQUENCY_DOMAIN_SSDs

Returns

No return value

Example

To unset the flag f on all the *FREQUENCY_DOMAIN_SSDs in model m:

```
FreqSSD.UnflagAll(m, f);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

FreqSSD object.

Return type

FreqSSD

Example

To check if FreqSSD property f.example is a parameter by using the FreqSSD.GetParameter() method:

```
if (f.ViewParameters().GetParameter(f.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for *FREQUENCY_DOMAIN_SSD with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The *FREQUENCY_DOMAIN_SSD object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for *FREQUENCY_DOMAIN_SSD f:

```
f.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for *FREQUENCY_DOMAIN_SSD f:

```
f.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for *FREQUENCY_DOMAIN_SSD. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for *FREQUENCY_DOMAIN_SSD f:

```
f.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this *FREQUENCY_DOMAIN_SSD.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for *FREQUENCY_DOMAIN_SSD f:

```
var xrefs = f.Xrefs();
```

---

toString()

Description

Creates a string containing the *FREQUENCY_DOMAIN_SSD data in keyword format. Note that this contains the keyword header and the keyword cards. See also FreqSSD.Keyword() and FreqSSD.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for *FREQUENCY_DOMAIN_SSD f in keyword format

```
var s = f.toString();
```

FreqVibration class

The FreqVibration class gives you access to *FREQUENCY_DOMAIN_RANDOM_VIBRATION keyword in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetAutoPsdLoadData(index*[integer]*)
- GetCrossPsdLoadData(index*[integer]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- GetInftgData(index*[integer]*)
- GetSNCurveData(index*[integer]*)
- Last(Model*[Model]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SetAutoPsdLoadData(index*[integer]*, data*[Array of data]*)
- SetCrossPsdLoadData(index*[integer]*, data*[Array of data]*)
- SetInftgData(index*[integer]*, filename*[string]*)
- SetSNCurveData(index*[integer]*, data*[Array of data]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)

Member functions

- AddAutoPsdLoadData()
- AddCrossPsdLoadData()
- AddInftgData()
- AddSNCurveData()
- AssociateComment(Comment*[Comment]*)
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- RemoveAutoPsdLoadData(index*[integer]*)
- RemoveCrossPsdLoadData(index*[integer]*)
- RemoveInftgData(index*[integer]*)
- RemoveSNCurveData(index*[integer]*)
- SetFlag(flag*[Flag]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()

- toString()

FreqVibration constants

Name	Description
FreqVibration.FATIGUE	FreqVibration option for *FREQUENCY_DOMAIN_RANDOM_VIBRATION_FATIGUE.
FreqVibration.VIBRATION	FreqVibration option for *FREQUENCY_DOMAIN_RANDOM_VIBRATION.

FreqVibration properties

Name	Type	Description
dampf	real	Modal damping coefficient.
dmpmas	real	Mass proportional damping constant in Rayleigh damping.
dmpstf	real	Stiffness proportional damping constant in Rayleigh damping.
dmptyp	integer	Tytpe of Damping.
exists (read only)	logical	true if *FREQUENCY_DOMAIN_RANDOM_VIBRATION exists, false if referred to but not defined.
fnmax	real	The maximum natural frequency in modal superposition method (optional).
fnmin	real	The minimum natural frequency in modal superposition method (optional).
icoarse	integer	Option for PSD curve coarsening.
include	integer	The Include file number that the *FREQUENCY_DOMAIN_RANDOM_VIBRATION is in.
inftg	integer	Flag for including initial damage ratio.
ipanelu	integer	Number of strips in U-direction (used only for vaflag = 5,6,7).
ipanelv	integer	Number of strips in V-direction (used only for vaflag = 5,6,7).
label (read only)	integer	The label the *FREQUENCY_DOMAIN_RANDOM_VIBRATION has in PRIMER.
lcdam	integer	Load Curve ID defining mode dependent modal damping coefficient.
lctyp	integer	Type of load curve defining modal damping coefficient.
ldflag	integer	Type of loading curves.
ldtyp	Integer	Excitation load type.
mdmax	integer	The last mode in modal superposition method (optional).
mdmin	integer	The first mode in modal superposition method (optional).
method	integer	Method for modal response analysis.
mftg	integer	Method for random fatigue analysis.
model (read only)	integer	The Model number that the *FREQUENCY_DOMAIN_RANDOM_VIBRATION is in.
napsd	integer	Number of auto PSD load definition.
ncpsd	integer	Number of cross PSD load definition.
nftg	integer	Field specifying the number of S-N curves to be defined.
option	constant	The *FREQUENCY_DOMAIN_RANDOM_VIBRATION option. Can be FreqVibration.VIBRATION, FreqVibration.FATIGUE.

pref	real	Reference pressure
restrm	integer	Restart option.
restrt	integer	Restart option.
strsf	real	Stress scale factor to accommodate different ordinates in S-N curve (not used if nftg = -999).
strtyp	integer	Stress type of S-N curve in fatigue analysis.
tcoarse	real	Tolerance for slope change percentage for removing intermediate points from PSD curve for icoarse = 2.
temper	real	Temperature.
texpos	real	Exposure time.
umlt	real	Multiplier for converting g to [length unit]/[time unit]^2 (used only for UNIT = -1).
unit	integer	Flag for acceleration unit conversion.
vaflag	integer	Loading type.
vapsd	integer	Flag for PSD output.
varms	integer	Flag for RMS output.

Detailed Description

The FreqVibration class allows you to create, modify, edit and manipulate *FREQUENCY_DOMAIN_RANDOM_VIBRATION. See the documentation below for more details.

Constructor

new FreqVibration(Model*[Model]*, option*[constant]*)

Description

Create a new FreqVibration object.

Arguments

- **Model** (Model)

Model that *FREQUENCY_DOMAIN_RANDOM_VIBRATION will be created in.

- **option** (constant)

Specify the type of *FREQUENCY_DOMAIN_RANDOM_VIBRATION. Can be FreqVibration.VIBRATION, FreqVibration.FATIGUE.

Returns

FreqVibration object

Return type

FreqVibration

Example

To create a new *FREQUENCY_DOMAIN_RANDOM_VIBRATION in model m, of type FATIGUE

```
var f = new FreqVibration(m, FreqVibration.FATIGUE);
```

Details of functions

AddAutoPsdLoadData()

Description

Allows user to add a new Auto PSD load card in *FREQUENCY_DOMAIN_RANDOM_VIBRATION.

The new card has uninitialised fields and should be updated by FreqVibration.SetAutoPsdLoadData().

Arguments

No arguments

Returns

Index of the new auto PSD load

Return type

integer

Example

To add a new auto PSD load data card in *FREQUENCY_DOMAIN_RANDOM_VIBRATION f:

```
f.AddAutoPsdLoadData();
```

AddCrossPsdLoadData()

Description

Allows user to add a new Cross PSD load card in *FREQUENCY_DOMAIN_RANDOM_VIBRATION.

The new card has uninitialised fields and should be updated by FreqVibration.SetCrossPsdLoadData().

Arguments

No arguments

Returns

Index of the new cross PSD load

Return type

integer

Example

To add a new cross PSD load data card in *FREQUENCY_DOMAIN_RANDOM_VIBRATION f:

```
f.AddCrossPsdLoadData();
```

AddInftgData()

Description

Allows user to add new Initial Damage cards in *FREQUENCY_DOMAIN_RANDOM_VIBRATION. This method is only applicable when option is FreqVibration.FATIGUE.

The new cards have uninitialised fields and should be updated by FreqVibration.SetInftgData().

Arguments

No arguments

Returns

Index of the new initial damage card

Return type

integer

Example

To add a new initial damage card in *FREQUENCY_DOMAIN_RANDOM_VIBRATION f:

```
f.AddInftgData();
```

AddSNCurveData()

Description

Allows user to add new S-N curve cards in *FREQUENCY_DOMAIN_RANDOM_VIBRATION. This method is only applicable when option is FreqVibration.FATIGUE.

The new cards have uninitialised fields and should be updated by FreqVibration.SetSNCurveData().

Arguments

No arguments

Returns

Index of the new S-N curve card

Return type

integer

Example

To add a new S-N curve card in *FREQUENCY_DOMAIN_RANDOM_VIBRATION f:

```
f.AddSNCurveData();
```

AssociateComment(Comment[*Comment*])

Description

Associates a comment with a *FREQUENCY_DOMAIN_RANDOM_VIBRATION.

Arguments

- **Comment** (Comment)

Comment that will be attached to the *FREQUENCY_DOMAIN_RANDOM_VIBRATION

Returns

No return value

Example

To associate comment c to the *FREQUENCY_DOMAIN_RANDOM_VIBRATION f:

```
f.AssociateComment(c);
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse *FREQUENCY_DOMAIN_RANDOM_VIBRATION f:

```
f.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the *FREQUENCY_DOMAIN_RANDOM_VIBRATION.

Arguments

- **flag** (Flag)

Flag to clear on the *FREQUENCY_DOMAIN_RANDOM_VIBRATION

Returns

No return value

Example

To clear flag f for *FREQUENCY_DOMAIN_RANDOM_VIBRATION f:

```
f.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the *FREQUENCY_DOMAIN_RANDOM_VIBRATION. The target include of the copied *FREQUENCY_DOMAIN_RANDOM_VIBRATION can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

FreqVibration object

Return type

FreqVibration

Example

To copy *FREQUENCY_DOMAIN_RANDOM_VIBRATION f into *FREQUENCY_DOMAIN_RANDOM_VIBRATION z:

```
var z = f.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a *FREQUENCY_DOMAIN_RANDOM_VIBRATION

Arguments

- **Model** (Model)

Model that the *FREQUENCY_DOMAIN_RANDOM_VIBRATION will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

FreqVibration object (or null if not made).

Return type

FreqVibration

Example

To start creating a *FREQUENCY_DOMAIN_RANDOM_VIBRATION in model m:

```
var f = FreqVibration.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a *FREQUENCY_DOMAIN_RANDOM_VIBRATION.

Arguments

- **Comment** (Comment)

Comment that will be detached from the *FREQUENCY_DOMAIN_RANDOM_VIBRATION

Returns

No return value

Example

To detach comment c from the *FREQUENCY_DOMAIN_RANDOM_VIBRATION f:

```
f.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit *FREQUENCY_DOMAIN_RANDOM_VIBRATION f:

```
f.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for *FREQUENCY_DOMAIN_RANDOM_VIBRATION with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The *FREQUENCY_DOMAIN_RANDOM_VIBRATION object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for *FREQUENCY_DOMAIN_RANDOM_VIBRATION f:

```
f.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for *FREQUENCY_DOMAIN_RANDOM_VIBRATION f:

```
f.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for *FREQUENCY_DOMAIN_RANDOM_VIBRATION. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for *FREQUENCY_DOMAIN_RANDOM_VIBRATION f:

```
f.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first *FREQUENCY_DOMAIN_RANDOM_VIBRATION in the model.

Arguments

- **Model** (Model)

Model to get first *FREQUENCY_DOMAIN_RANDOM_VIBRATION in

Returns

FreqVibration object (or null if there are no *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs in the model).

Return type

FreqVibration

Example

To get the first *FREQUENCY_DOMAIN_RANDOM_VIBRATION in model m:

```
var f = FreqVibration.First(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs will be flagged in

- **flag** (Flag)

Flag to set on the *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs

Returns

No return value

Example

To flag all of the *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs with flag f in model m:

```
FreqVibration.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the *FREQUENCY_DOMAIN_RANDOM_VIBRATION is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the *FREQUENCY_DOMAIN_RANDOM_VIBRATION

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if *FREQUENCY_DOMAIN_RANDOM_VIBRATION f has flag f set on it:

```
if (f.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each *FREQUENCY_DOMAIN_RANDOM_VIBRATION in the model.
Note that ForEach has been designed to make looping over *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs as fast as possible and so has some limitations.
Firstly, a single temporary FreqVibration object is created and on each function call it is updated with the current *FREQUENCY_DOMAIN_RANDOM_VIBRATION data. This means that you should not try to store the FreqVibration object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs are in

- **func** (function)

Function to call for each *FREQUENCY_DOMAIN_RANDOM_VIBRATION

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs in model m:

```
FreqVibration.ForEach(m, test);
function test(f)
{
// f is FreqVibration object
}
```

To call function test for all of the *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs in model m with optional object:

```
var data = { x:0, y:0 };
FreqVibration.ForEach(m, test, data);
function test(f, extra)
{
// f is FreqVibration object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of FreqVibration objects or properties for all of the *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs in a model in PRIMER. If the optional property argument is not given then an array of FreqVibration objects is returned. If the property argument is given, that property value for each *FREQUENCY_DOMAIN_RANDOM_VIBRATION is returned in the array instead of a FreqVibration object

Arguments

- **Model** (Model)

Model to get *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs from

- **property (optional)** (string)

Name for property to get for all *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs in the model

Returns

Array of FreqVibration objects or properties

Return type

Array

Example

To make an array of FreqVibration objects for all of the *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs in model m:

```
var a = FreqVibration.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each *FREQUENCY_DOMAIN_RANDOM_VIBRATION in model m:

```
var a = FreqVibration.GetAll(m, 'foo');
```

GetAutoPsdLoadData(index*[integer]*) [static]

Description

Returns the Auto PSD load data for a specific Auto PSD Load definition as an array. For each Auto PSD load definition there will be 8 values. There are napsd Auto PSD load definitions.

Arguments

- **index** (integer)

Index you want the Auto PSD load data for. **Note that indices start at 0**.

Returns

An array containing the Auto PSD load data (values: sid[integer], stype[integer], dof[integer], ldpsd[integer], ldvel[integer], ldflw[integer], ldspn[integer], cid[integer]). The array length will be 8.

Return type

Number

Example

To get the data for the 3rd Auto PSD load data for *FREQUENCY_DOMAIN_RANDOM_VIBRATION f:

```
var apsd_data = f.GetAutoPsdLoadData(2);
```

GetComments()

Description

Extracts the comments associated to a *FREQUENCY_DOMAIN_RANDOM_VIBRATION.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the *FREQUENCY_DOMAIN_RANDOM_VIBRATION f:

```
var comm_array = f.GetComments();
```

GetCrossPsdLoadData(index*[integer]*) [static]

Description

Returns the Cross PSD load data for a specific Cross PSD Load definition as an array. For each Cross PSD load definition there will be 5 values. There are ncpsd Cross PSD load definitions.

Arguments

- **index** (integer)

Index you want the Cross PSD load data for. **Note that indices start at 0**.

Returns

An array containing the Cross PSD load data (values: load_i[integer], load_j[integer], lctyp2[integer], ldpsd1[integer], ldpsd2[integer]). The array length will be 5.

Return type

Number

Example

To get the data for the 3rd Cross PSD load data for *FREQUENCY_DOMAIN_RANDOM_VIBRATION f:

```
var cpsd_data = f.GetCrossPsdLoadData(2);
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of FreqVibration objects for all of the flagged *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs in a model in PRIMER If the optional property argument is not given then an array of FreqVibration objects is returned. If the property argument is given, then that property value for each *FREQUENCY_DOMAIN_RANDOM_VIBRATION is returned in the array instead of a FreqVibration object

Arguments

- **Model** (Model)

Model to get *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs from

- **flag** (Flag)

Flag set on the *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs in the model

Returns

Array of FreqVibration objects or properties

Return type

Array

Example

To make an array of FreqVibration objects for all of the *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs in model m flagged with f:

```
var f = FreqVibration.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs in model m flagged with f:

```
var a = FreqVibration.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the FreqVibration object for a *FREQUENCY_DOMAIN_RANDOM_VIBRATION ID.

Arguments

- **Model** (Model)

Model to find the *FREQUENCY_DOMAIN_RANDOM_VIBRATION in

- **number** (integer)

number of the *FREQUENCY_DOMAIN_RANDOM_VIBRATION you want the FreqVibration object for

Returns

FreqVibration object (or null if *FREQUENCY_DOMAIN_RANDOM_VIBRATION does not exist).

Return type

FreqVibration

Example

To get the FreqVibration object for *FREQUENCY_DOMAIN_RANDOM_VIBRATION 100 in model m

```
var f = FreqVibration.GetFromID(m, 100);
```

GetInftgData(index*[integer]*) [static]

Description

Returns the path and filename of a binary database for fatigue information from a specific initial damage card. There are inftg filenames.

This method is only applicable when option is FreqVibration.FATIGUE.

Arguments

- **index** (integer)

Index of an initial damage card that you want the filename from. **Note that indices start at 0**.

Returns

Return value from an initial damage card (values: filename[string])

Return type

String

Example

To get the filename from the 2nd initial damage card for *FREQUENCY_DOMAIN_RANDOM_VIBRATION f:

```
var fdata = f.GetInftgData(1);
```

GetParameter(prop*[string]*)

Description

Checks if a FreqVibration property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the FreqVibration.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

*FREQUENCY_DOMAIN_RANDOM_VIBRATION property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if FreqVibration property f.example is a parameter:

```
Options.property_parameter_names = true;
if (f.GetParameter(f.example) ) do_something...
Options.property_parameter_names = false;
```

To check if FreqVibration property f.example is a parameter by using the GetParameter method:

```
if (f.ViewParameters().GetParameter(f.example) ) do_something...
```

GetSNCurveData(index*[integer]*) [static]

Description

Returns the data of a specific zone for fatigue analysis as an array. For each zone there will be 8 values. There are nftg zone definitions for fatigue analysis.

This method is only applicable when option is FreqVibration.FATIGUE.

Arguments

- **index** (integer)

Index you want the zone data for. **Note that indices start at 0**.

Returns

An array containing the zone data (values: pid[integer], lcid[integer], ptype[integer], ltype[integer], a[real], b[real], sthres[real], snlimt[integer]).

The array length will be 8.

Return type

Number

Example

To get the data for the 4th zone for *FREQUENCY_DOMAIN_RANDOM_VIBRATION f:

```
var sndata = f.GetSNCurveData(3);
```

Keyword()

Description

Returns the keyword for this *FREQUENCY_DOMAIN_RANDOM_VIBRATION. **Note that a carriage return is not added**. See also FreqVibration.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for *FREQUENCY_DOMAIN_RANDOM_VIBRATION f:

```
var key = f.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the *FREQUENCY_DOMAIN_RANDOM_VIBRATION. **Note that a carriage return is not added**. See also FreqVibration.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for *FREQUENCY_DOMAIN_RANDOM_VIBRATION f:

```
var cards = f.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last *FREQUENCY_DOMAIN_RANDOM_VIBRATION in the model.

Arguments

- **Model** (Model)

Model to get last *FREQUENCY_DOMAIN_RANDOM_VIBRATION in

Returns

FreqVibration object (or null if there are no *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs in the model).

Return type

FreqVibration

Example

To get the last *FREQUENCY_DOMAIN_RANDOM_VIBRATION in model m:

```
var f = FreqVibration.Last(m);
```

Next()

Description

Returns the next *FREQUENCY_DOMAIN_RANDOM_VIBRATION in the model.

Arguments

No arguments

Returns

FreqVibration object (or null if there are no more *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs in the model).

Return type

FreqVibration

Example

To get the *FREQUENCY_DOMAIN_RANDOM_VIBRATION in model m after *FREQUENCY_DOMAIN_RANDOM_VIBRATION f:

```
var f = f.Next();
```

Previous()

Description

Returns the previous *FREQUENCY_DOMAIN_RANDOM_VIBRATION in the model.

Arguments

No arguments

Returns

FreqVibration object (or null if there are no more *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs in the model).

Return type

FreqVibration

Example

To get the *FREQUENCY_DOMAIN_RANDOM_VIBRATION in model m before *FREQUENCY_DOMAIN_RANDOM_VIBRATION f:

```
var f = f.Previous();
```

RemoveAutoPsdLoadData(index*[integer]*)

Description

Allows user to remove a specified Auto PSD load card in *FREQUENCY_DOMAIN_RANDOM_VIBRATION.

Arguments

- **index** (integer)

Index of the auto PSD load card you want to remove. **Note that indices start at 0**.

Returns

No return value

Example

To remove first load card in *FREQUENCY_DOMAIN_RANDOM_VIBRATION f:

```
f.RemoveAutoPsdLoadData(0);
```

RemoveCrossPsdLoadData(index*[integer]*)

Description

Allows user to remove a specified Cross PSD load card in *FREQUENCY_DOMAIN_RANDOM_VIBRATION.

Arguments

- **index** (integer)

Index of the cross PSD load card you want to remove. **Note that indices start at 0**.

Returns

No return value

Example

To remove third load card in *FREQUENCY_DOMAIN_RANDOM_VIBRATION f:

```
f.RemoveCrossPsdLoadData(2);
```

RemoveInftgData(index*[integer]*)

Description

Allows user to remove a specified Initial Damage card in *FREQUENCY_DOMAIN_RANDOM_VIBRATION.
This method is only applicable when option is FreqVibration.FATIGUE.

Arguments

- **index** (integer)

Index of the Initrial Damage card you want to remove. **Note that indices start at 0**.

Returns

No return value

Example

To remove second Initial Damage card in *FREQUENCY_DOMAIN_RANDOM_VIBRATION f:

```
f.RemoveInftgData(1);
```

RemoveSNCurveData(index*[integer]*)

Description

Allows user to remove a specified S-N curve card in *FREQUENCY_DOMAIN_RANDOM_VIBRATION.
This method is only applicable when option is FreqVibration.FATIGUE.

Arguments

- **index** (integer)

Index of the S-N curve card you want to remove. **Note that indices start at 0**.

Returns

No return value

Example

To remove second S-N curve card in *FREQUENCY_DOMAIN_RANDOM_VIBRATION f:

```
f.RemoveSNCurveData(1);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs from that model can be selected. If the argument is a Flag then only *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs selected or null if menu cancelled

Return type

Number

Example

To select *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs from model m, flagging those selected with flag f, giving the prompt 'Select *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs':

```
FreqVibration.Select(f, 'Select *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs', m);
```

To select *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs, flagging those selected with flag f but limiting selection to *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs flagged with flag l, giving the prompt 'Select *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs':

```
FreqVibration.Select(f, 'Select *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs', l);
```

SetAutoPsdLoadData(index*[integer]*, data*[Array of data]*) [static]

Description

Set the data for a specific Auto PSD load card. For each Auto PSD load card there will be 8 values. There are napsd Auto PSD load cards.

Arguments

- **index** (integer)

Index you want to set Auto PSD load data for. **Note that indices start at 0**.

- **data** (Array of data)

An array containing the Auto PSD load data (values: sid[integer], stype[integer], dof[integer], ldpsd[integer], ldvel[integer], ldflw[integer], ldspn[integer], cid[integer]). The array length should be 8.

Returns

No return value.

Example

To set the 3rd Auto PSD load data for *FREQUENCY_DOMAIN_RANDOM_VIBRATION f to the values in array adata:

```
f.SetAutoPsdLoadData(2, adata);
```

SetCrossPsdLoadData(index*[integer]*, data*[Array of data]*) [static]

Description

Set the data for a specific Cross PSD load card. For each Cross PSD load card there will be 5 values. There are ncpsd Cross PSD load cards.

Arguments

- **index** (integer)

Index you want to set Cross PSD load data for. **Note that indices start at 0**.

- **data** (Array of data)

An array containing the Cross PSD load data (values: load_i[integer], load_j[integer], lctyp2[integer], ldpsd1[integer], ldpsd2[integer]). The array length should be 5.

Returns

No return value.

Example

To set the 4th Cross PSD load data for *FREQUENCY_DOMAIN_RANDOM_VIBRATION f to the values in array cdata:

```
f.SetCrossPsdLoadData(2, cdata);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the *FREQUENCY_DOMAIN_RANDOM_VIBRATION.

Arguments

- **flag** (Flag)

Flag to set on the *FREQUENCY_DOMAIN_RANDOM_VIBRATION

Returns

No return value

Example

To set flag f for *FREQUENCY_DOMAIN_RANDOM_VIBRATION f:

```
f.SetFlag(f);
```

SetInftgData(index*[integer]*, filename*[string]*) [static]

Description

Set the filename data for an existing binary database for fatigue infromation for a specific initial damage card. There are inftg filenames.

This method is only applicable when option is FreqVibration.FATIGUE.

Arguments

- **index** (integer)

Index of an initial damage card that you want the filename for. **Note that indices start at 0**.

- **filename** (string)

Path and name of existing binary database fro fatigue information.

Returns

No return value.

Example

To set the file name for the 2nd initial damage card of *FREQUENCY_DOMAIN_RANDOM_VIBRATION f to have filename "d3ftg":

```
f.SetInftgData(1, "d3ftg");
```

SetSNCurveData(index*[integer]*, data*[Array of data]*) [static]

Description

Set the data for a specific zone for fatigue analysis. For each zone there will be 8 values. There are nftg zone definitions for fatigue analysis. This method is only applicable when option is FreqVibration.FATIGUE.

Arguments

- **index** (integer)

Index you want to set the fatigue analysis zone data for. **Note that indices start at 0**.

- **data** (Array of data)

An array containing the zone data (values: pid[integer], lcid[integer], ptype[integer], ltype[integer], a[float], b[float], sthres[float], snlimt[integer]). The array length will be 8.

Returns

No return value.

Example

To set the data for 4th fatigue analysis zone in *FREQUENCY_DOMAIN_RANDOM_VIBRATION f to the values in array sndata:

```
f.SetSNCurveData(3, sndata);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs should be counted. If false or omitted referenced but undefined *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs will also be included in the total.

Returns

number of *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs

Return type

Number

Example

To get the total number of *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs in model m:

```
var total = FreqVibration.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs will be unset in

- **flag** (Flag)

Flag to unset on the *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs

Returns

No return value

Example

To unset the flag f on all the *FREQUENCY_DOMAIN_RANDOM_VIBRATIONs in model m:

```
FreqVibration.UnflagAll(m, f);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

FreqVibration object.

Return type

FreqVibration

Example

To check if FreqVibration property f.example is a parameter by using the FreqVibration.GetParameter() method:

```
if (f.ViewParameters().GetParameter(f.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for *FREQUENCY_DOMAIN_RANDOM_VIBRATION with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The *FREQUENCY_DOMAIN_RANDOM_VIBRATION object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for *FREQUENCY_DOMAIN_RANDOM_VIBRATION f:

```
f.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for *FREQUENCY_DOMAIN_RANDOM_VIBRATION f:

```
f.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for *FREQUENCY_DOMAIN_RANDOM_VIBRATION. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for *FREQUENCY_DOMAIN_RANDOM_VIBRATION f:

```
f.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this *FREQUENCY_DOMAIN_RANDOM_VIBRATION.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for *FREQUENCY_DOMAIN_RANDOM_VIBRATION f:

```
var xrefs = f.Xrefs();
```

---

toString()

Description

Creates a string containing the *FREQUENCY_DOMAIN_RANDOM_VIBRATION data in keyword format. Note that this contains the keyword header and the keyword cards. See also FreqVibration.Keyword() and FreqVibration.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for *FREQUENCY_DOMAIN_RANDOM_VIBRATION f in keyword format

```
var s = f.toString();
```

Hourglass class

The Hourglass class gives you access to hourglass cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

Hourglass properties

Name	Type	Description
exists (read only)	logical	true if hourglass exists, false if referred to but not defined.
hgid	integer or string	Hourglass number or character label. Also see the label property which is an alternative name for this.

ibq	integer	Bulk viscosity type
ihq	integer	Hourglass control type
include	integer	The Include file number that the hourglass is in.
label	integer or string	Hourglass number or character label. Also see the hgid property which is an alternative name for this.
model (read only)	integer	The Model number that the hourglass is in.
q1	real	Quadratic bulk viscosity coefficient
q2	real	Linear bulk viscosity coefficient
qb	real	Coefficient for shell bending
qm	real	Hourglass coefficient
qw	real	Coefficient for shell warping
title	string	Title for hourglass
vdc	real	Viscous damping coefficient

Detailed Description

The Hourglass class allows you to create, modify, edit and manipulate hourglass cards. See the documentation below for more details.

Constructor

new Hourglass(Model*[Model]*, hgid*[integer or string]*, title (optional)*[string]*)

Description

Create a new Hourglass object.

Arguments

- **Model** (Model)

Model that hourglass will be created in

- **hgid** (integer or string)

Hourglass number or character label

- **title (optional)** (string)

Title for the hourglass

Returns

Hourglass object

Return type

Hourglass

Example

To create a new hourglass in model m with label 100:

```
var h = new Hourglass(m, 100);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a hourglass.

Arguments

- **Comment** (Comment)

Comment that will be attached to the hourglass

Returns

No return value

Example

To associate comment c to the hourglass n:

```
n.AssociateComment(c);
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse hourglass n:

```
n.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the hourglass.

Arguments

- **flag** (Flag)

Flag to clear on the hourglass

Returns

No return value

Example

To clear flag f for hourglass n:

```
n.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the hourglass. The target include of the copied hourglass can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

Hourglass object

Return type

Hourglass

Example

To copy hourglass n into hourglass z:

```
var z = n.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a hourglass

Arguments

- **Model** (Model)

Model that the hourglass will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

Hourglass object (or null if not made).

Return type

Hourglass

Example

To start creating a hourglass in model m:

```
var n = Hourglass.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a hourglass.

Arguments

- **Comment** (Comment)

Comment that will be detached from the hourglass

Returns

No return value

Example

To detach comment c from the hourglass n:

```
n.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit hourglass n:

```
n.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for hourglass with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description

autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The hourglass object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for hourglass n:

```
n.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for hourglass n:

```
n.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for hourglass. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for hourglass n:

```
n.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first hourglass in the model.

Arguments

- **Model** (Model)

Model to get first hourglass in

Returns

Hourglass object (or null if there are no hourglasses in the model).

Return type

Hourglass

Example

To get the first hourglass in model m:

```
var n = Hourglass.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free hourglass label in the model. Also see Hourglass.LastFreeLabel(), Hourglass.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free hourglass label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

Hourglass label.

Return type

Number

Example

To get the first free hourglass label in model m:

```
var label = Hourglass.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the hourglasses in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all hourglasses will be flagged in

- **flag** (Flag)

Flag to set on the hourglasses

Returns

No return value

Example

To flag all of the hourglasses with flag f in model m:

```
Hourglass.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the hourglass is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the hourglass

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if hourglass n has flag f set on it:

```
if (n.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each hourglass in the model.
Note that ForEach has been designed to make looping over hourglasses as fast as possible and so has some limitations.
Firstly, a single temporary Hourglass object is created and on each function call it is updated with the current hourglass data. This means that you should not try to store the Hourglass object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new hourglasses inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all hourglasses are in

- **func** (function)

Function to call for each hourglass

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the hourglasses in model m:

```
Hourglass.ForEach(m, test);
function test(n)
{
// n is Hourglass object
}
```

To call function test for all of the hourglasses in model m with optional object:

```
var data = { x:0, y:0 };
Hourglass.ForEach(m, test, data);
function test(n, extra)
{
// n is Hourglass object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Hourglass objects or properties for all of the hourglasses in a model in PRIMER. If the optional property argument is not given then an array of Hourglass objects is returned. If the property argument is given, that property value for each hourglass is returned in the array instead of a Hourglass object

Arguments

- **Model** (Model)

Model to get hourglasses from

- **property (optional)** (string)

Name for property to get for all hourglasses in the model

Returns

Array of Hourglass objects or properties

Return type

Array

Example

To make an array of Hourglass objects for all of the hourglasses in model m:

```
var a = Hourglass.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each hourglass in model m:

```
var a = Hourglass.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a hourglass.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the hourglass n:

```
var comm_array = n.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of Hourglass objects for all of the flagged hourglasses in a model in PRIMER If the optional property argument is not given then an array of Hourglass objects is returned. If the property argument is given, then that property value for each hourglass is returned in the array instead of a Hourglass object

Arguments

- **Model** (Model)

Model to get hourglasses from

- **flag** (Flag)

Flag set on the hourglasses that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged hourglasses in the model

Returns

Array of Hourglass objects or properties

Return type

Array

Example

To make an array of Hourglass objects for all of the hourglasses in model m flagged with f:

```
var n = Hourglass.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the hourglasses in model m flagged with f:

```
var a = Hourglass.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the Hourglass object for a hourglass ID.

Arguments

- **Model** (Model)

Model to find the hourglass in

- **number** (integer)

number of the hourglass you want the Hourglass object for

Returns

Hourglass object (or null if hourglass does not exist).

Return type

Hourglass

Example

To get the Hourglass object for hourglass 100 in model m

```
var n = Hourglass.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a Hourglass property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the Hourglass.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

hourglass property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if Hourglass property n.example is a parameter:

```
Options.property_parameter_names = true;
if (n.GetParameter(n.example) ) do_something...
Options.property_parameter_names = false;
```

To check if Hourglass property n.example is a parameter by using the GetParameter method:

```
if (n.ViewParameters().GetParameter(n.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this hourglass (*HOURGLASS). **Note that a carriage return is not added**. See also Hourglass.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for hourglass h:

```
var key = h.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the hourglass. **Note that a carriage return is not added**. See also Hourglass.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for hourglass h:

```
var cards = h.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last hourglass in the model.

Arguments

- **Model** (Model)

Model to get last hourglass in

Returns

Hourglass object (or null if there are no hourglasses in the model).

Return type

Hourglass

Example

To get the last hourglass in model m:

```
var n = Hourglass.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free hourglass label in the model. Also see Hourglass.FirstFreeLabel(), Hourglass.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free hourglass label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

Hourglass label.

Return type

Number

Example

To get the last free hourglass label in model m:

```
var label = Hourglass.LastFreeLabel(m);
```

Next()

Description

Returns the next hourglass in the model.

Arguments

No arguments

Returns

Hourglass object (or null if there are no more hourglasses in the model).

Return type

Hourglass

Example

To get the hourglass in model m after hourglass n:

```
var n = n.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) hourglass label in the model. Also see Hourglass.FirstFreeLabel(), Hourglass.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free hourglass label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

Hourglass label.

Return type

Number

Example

To get the next free hourglass label in model m:

```
var label = Hourglass.NextFreeLabel(m);
```

Previous()

Description

Returns the previous hourglass in the model.

Arguments

No arguments

Returns

Hourglass object (or null if there are no more hourglasses in the model).

Return type

Hourglass

Example

To get the hourglass in model m before hourglass n:

```
var n = n.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the hourglasses in the model.

Arguments

- **Model** (Model)

Model that all hourglasses will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the hourglasses in model m, from 1000000:

```
Hourglass.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged hourglasses in the model.

Arguments

- **Model** (Model)

Model that all the flagged hourglasses will be renumbered in

- **flag** (Flag)

Flag set on the hourglasses that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the hourglasses in model m flagged with f, from 1000000:

```
Hourglass.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select hourglasses using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting hourglasses

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only hourglasses from that model can be selected. If the argument is a Flag then only hourglasses that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any hourglasses can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of hourglasses selected or null if menu cancelled

Return type

Number

Example

To select hourglasses from model m, flagging those selected with flag f, giving the prompt 'Select hourglasses':

```
Hourglass.Select(f, 'Select hourglasses', m);
```

To select hourglasses, flagging those selected with flag f but limiting selection to hourglasses flagged with flag l, giving the prompt 'Select hourglasses':

```
Hourglass.Select(f, 'Select hourglasses', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the hourglass.

Arguments

- **flag** (Flag)

Flag to set on the hourglass

Returns

No return value

Example

To set flag f for hourglass n:

```
n.SetFlag(f);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of hourglasses in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing hourglasses should be counted. If false or omitted referenced but undefined hourglasses will also be included in the total.

Returns

number of hourglasses

Return type

Number

Example

To get the total number of hourglasses in model m:

```
var total = Hourglass.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the hourglasses in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all hourglasses will be unset in

- **flag** (Flag)

Flag to unset on the hourglasses

Returns

No return value

Example

To unset the flag f on all the hourglasses in model m:

```
Hourglass.UnflagAll(m, f);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

Hourglass object.

Return type

Hourglass

Example

To check if Hourglass property n.example is a parameter by using the Hourglass.GetParameter() method:

```
if (n.ViewParameters().GetParameter(n.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for hourglass with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The hourglass object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for hourglass n:

```
n.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for hourglass n:

```
n.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for hourglass. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for hourglass n:

```
n.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this hourglass.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for hourglass n:

```
var xrefs = n.Xrefs();
```

---

toString()

Description

Creates a string containing the hourglass data in keyword format. Note that this contains the keyword header and the keyword cards. See also Hourglass.Keyword() and Hourglass.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for hourglass h in keyword format

```
var s = h.toString();
```

IGA1DBrep class

The IGA1DBrep class gives you access to IGA 1d brep cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetData(index*[integer]*)
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- RemoveData(index*[integer]*)
- SetData(index*[integer]*, eid*[integer]*)
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)

- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

IGA1DBrep properties

Name	Type	Description
brid	integer	IGA1DBrep number. Also see the label property which is an alternative name for this.
entries (read only)	integer	Number of Parametric Edge IDs (eid) added in IGA_1D_BREP. Can be used as index to add a new entry.
exists (read only)	logical	true if IGA 1D Brep exists, false if referred to but not defined.
include	integer	The Include file number that the IGA 1D Brep is in.
label	integer	IGA1DBrep number. Also see the brid property which is an alternative name for this.
model (read only)	integer	The Model number that the IGA 1D Brep is in.

Detailed Description

The IGA1DBrep class allows you to create, modify, edit and manipulate IGA 1d brep cards. See the documentation below for more details.

Constructor

new IGA1DBrep(Model*[Model]*, details*[object]*)

Description

Create a new IGA1DBrep object.

Arguments

- **Model** (Model)

Model that IGA 1d brep will be created in

- **details** (object)

Details for creating the IGA1DBrep

Object has the following properties:

Name	Type	Description
brid	integer	ID of the IGA 1D Brep.

Returns

IGA1DBrep object

Return type

IGA1DBrep

Example

To create a new IGA 1d brep in model m of id 10:

```
var ig1b = new IGA1DBrep(m, {brid:10});
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a IGA 1D Brep.

Arguments

- **Comment** (Comment)

Comment that will be attached to the IGA 1D Brep

Returns

No return value

Example

To associate comment c to the IGA 1D Brep ig1b:

```
ig1b.AssociateComment(c);
```

Blank()

Description

Blanks the IGA 1D Brep

Arguments

No arguments

Returns

No return value

Example

To blank IGA 1D Brep ig1b:

```
ig1b.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the IGA 1D Breps in the model.

Arguments

- **Model** (Model)

Model that all IGA 1D Breps will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the IGA 1D Breps in model m:

```
IGA1DBrep.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged IGA 1D Breps in the model.

Arguments

- **Model** (Model)

Model that all the flagged IGA 1D Breps will be blanked in

- **flag** (Flag)

Flag set on the IGA 1D Breps that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the IGA 1D Breps in model m flagged with f:

```
IGA1DBrep.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the IGA 1D Brep is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if IGA 1D Brep ig1b is blanked:

```
if (ig1b.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse IGA 1D Brep ig1b:

```
ig1b.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the IGA 1D Brep.

Arguments

- **flag** (Flag)

Flag to clear on the IGA 1D Brep

Returns

No return value

Example

To clear flag f for IGA 1D Brep ig1b:

```
ig1b.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the IGA 1D Brep. The target include of the copied IGA 1D Brep can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

IGA1DBrep object

Return type

IGA1DBrep

Example

To copy IGA 1D Brep ig1b into IGA 1D Brep z:

```
var z = ig1b.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a IGA 1D Brep

Arguments

- **Model** (Model)

Model that the IGA 1D Brep will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

IGA1DBrep object (or null if not made).

Return type

IGA1DBrep

Example

To start creating a IGA 1D Brep in model m:

```
var ig1b = IGA1DBrep.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a IGA 1D Brep.

Arguments

- **Comment** (Comment)

Comment that will be detached from the IGA 1D Brep

Returns

No return value

Example

To detach comment c from the IGA 1D Brep ig1b:

```
ig1b.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit IGA 1D Brep ig1b:

```
ig1b.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for IGA 1D Brep with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA 1D Brep object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for IGA 1D Brep ig1b:

```
ig1b.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for IGA 1D Brep ig1b:

```
ig1b.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for IGA 1D Brep. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for IGA 1D Brep ig1b:

```
ig1b.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first IGA 1D Brep in the model.

Arguments

- **Model** (Model)

Model to get first IGA 1D Brep in

Returns

IGA1DBrep object (or null if there are no IGA 1D Breps in the model).

Return type

IGA1DBrep

Example

To get the first IGA 1D Brep in model m:

```
var ig1b = IGA1DBrep.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free IGA 1D Brep label in the model. Also see IGA1DBrep.LastFreeLabel(), IGA1DBrep.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free IGA 1D Brep label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

IGA1DBrep label.

Return type

Number

Example

To get the first free IGA 1D Brep label in model m:

```
var label = IGA1DBrep.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the IGA 1D Breps in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all IGA 1D Breps will be flagged in

- **flag** (Flag)

Flag to set on the IGA 1D Breps

Returns

No return value

Example

To flag all of the IGA 1D Breps with flag f in model m:

```
IGA1DBrep.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the IGA 1D Brep is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the IGA 1D Brep

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if IGA 1D Brep ig1b has flag f set on it:

```
if (ig1b.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each IGA 1D Brep in the model.
Note that ForEach has been designed to make looping over IGA 1D Breps as fast as possible and so has some limitations.
Firstly, a single temporary IGA1DBrep object is created and on each function call it is updated with the current IGA 1D Brep data. This means that you should not try to store the IGA1DBrep object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new IGA 1D Breps inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all IGA 1D Breps are in

- **func** (function)

Function to call for each IGA 1D Brep

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the IGA 1D Breps in model m:

```
IGA1DBrep.ForEach(m, test);
function test(ig1b)
{
// ig1b is IGA1DBrep object
}
```

To call function test for all of the IGA 1D Breps in model m with optional object:

```
var data = { x:0, y:0 };
IGA1DBrep.ForEach(m, test, data);
function test(ig1b, extra)
{
// ig1b is IGA1DBrep object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of IGA1DBrep objects or properties for all of the IGA 1D Breps in a model in PRIMER. If the optional property argument is not given then an array of IGA1DBrep objects is returned. If the property argument is given, that property value for each IGA 1D Brep is returned in the array instead of a IGA1DBrep object

Arguments

- **Model** (Model)

Model to get IGA 1D Breps from

- **property (optional)** (string)

Name for property to get for all IGA 1D Breps in the model

Returns

Array of IGA1DBrep objects or properties

Return type

Array

Example

To make an array of IGA1DBrep objects for all of the IGA 1D Breps in model m:

```
var a = IGA1DBrep.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each IGA 1D Brep in model m:

```
var a = IGA1DBrep.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a IGA 1D Brep.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the IGA 1D Brep ig1b:

```
var comm_array = ig1b.GetComments();
```

GetData(index*[integer]*)

Description

Returns the data for eid in *IGA_1D_BREP.

Arguments

- **index** (integer)

Index you want the data for. **Note that indices start at 0**.

Returns

The ID of parametric edge, *IGA_EDGE_UVW.

Return type

Number

Example

To get the 3rd data entry in IGA 1d brep ig1b:

```
var data = ig1b.GetData(2);
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of IGA1DBrep objects for all of the flagged IGA 1D Breps in a model in PRIMER If the optional property argument is not given then an array of IGA1DBrep objects is returned. If the property argument is given, then that property value for each IGA 1D Brep is returned in the array instead of a IGA1DBrep object

Arguments

- **Model** (Model)

Model to get IGA 1D Breps from

- **flag** (Flag)

Flag set on the IGA 1D Breps that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged IGA 1D Breps in the model

Returns

Array of IGA1DBrep objects or properties

Return type

Array

Example

To make an array of IGA1DBrep objects for all of the IGA 1D Breps in model m flagged with f:

```
var ig1b = IGA1DBrep.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the IGA 1D Breps in model m flagged with f:

```
var a = IGA1DBrep.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the IGA1DBrep object for a IGA 1D Brep ID.

Arguments

- **Model** (Model)

Model to find the IGA 1D Brep in

- **number** (integer)

number of the IGA 1D Brep you want the IGA1DBrep object for

Returns

IGA1DBrep object (or null if IGA 1D Brep does not exist).

Return type

IGA1DBrep

Example

To get the IGA1DBrep object for IGA 1D Brep 100 in model m

```
var ig1b = IGA1DBrep.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a IGA1DBrep property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the IGA1DBrep.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

IGA 1D Brep property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if IGA1DBrep property ig1b.example is a parameter:

```
Options.property_parameter_names = true;
if (ig1b.GetParameter(ig1b.example) ) do_something...
Options.property_parameter_names = false;
```

To check if IGA1DBrep property ig1b.example is a parameter by using the GetParameter method:

```
if (ig1b.ViewParameters().GetParameter(ig1b.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this IGA 1d brep (*IGA_1D_BREP). **Note that a carriage return is not added**. See also IGA1DBrep.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for IGA 1d brep ig1b:

```
var key = ig1b.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the IGA 1d brep. **Note that a carriage return is not added**. See also IGA1DBrep.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for IGA 1d brep ig1b:

```
var cards = ig1b.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last IGA 1D Brep in the model.

Arguments

- **Model** (Model)

Model to get last IGA 1D Brep in

Returns

IGA1DBrep object (or null if there are no IGA 1D Breps in the model).

Return type

IGA1DBrep

Example

To get the last IGA 1D Brep in model m:

```
var ig1b = IGA1DBrep.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free IGA 1D Brep label in the model. Also see IGA1DBrep.FirstFreeLabel(), IGA1DBrep.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free IGA 1D Brep label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

IGA1DBrep label.

Return type

Number

Example

To get the last free IGA 1D Brep label in model m:

```
var label = IGA1DBrep.LastFreeLabel(m);
```

Next()

Description

Returns the next IGA 1D Brep in the model.

Arguments

No arguments

Returns

IGA1DBrep object (or null if there are no more IGA 1D Breps in the model).

Return type

IGA1DBrep

Example

To get the IGA 1D Brep in model m after IGA 1D Brep ig1b:

```
var ig1b = ig1b.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) IGA 1D Brep label in the model. Also see IGA1DBrep.FirstFreeLabel(), IGA1DBrep.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free IGA 1D Brep label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

IGA1DBrep label.

Return type

Number

Example

To get the next free IGA 1D Brep label in model m:

```
var label = IGA1DBrep.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a IGA 1D Brep.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only IGA 1D Breps from that model can be picked. If the argument is a Flag then only IGA 1D Breps that are flagged with *limit* can be selected. If omitted, or null, any IGA 1D Breps from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

IGA1DBrep object (or null if not picked)

Return type

IGA1DBrep

Example

To pick a IGA 1D Brep from model m giving the prompt 'Pick IGA 1D Brep from screen':

```
var ig1b = IGA1DBrep.Pick('Pick IGA 1D Brep from screen', m);
```

Previous()

Description

Returns the previous IGA 1D Brep in the model.

Arguments

No arguments

Returns

IGA1DBrep object (or null if there are no more IGA 1D Breps in the model).

Return type

IGA1DBrep

Example

To get the IGA 1D Brep in model m before IGA 1D Brep ig1b:

```
var ig1b = ig1b.Previous();
```

RemoveData(index*[integer]*)

Description

Removes eid for an index in *IGA_1D_BREP.

Arguments

- **index** (integer)

The index you want to delete eid for. **Note that indices start at 0, not 1**.

Returns

No return value.

Example

To delete the eid for the 3rd index for *IGA_1D_BREP in model m:

```
if (ig1b.entries >= 3)
{
    ig1b.RemoveData(2);
}
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the IGA 1D Breps in the model.

Arguments

- **Model** (Model)

Model that all IGA 1D Breps will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the IGA 1D Breps in model m, from 1000000:

```
IGA1DBrep.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged IGA 1D Breps in the model.

Arguments

- **Model** (Model)

Model that all the flagged IGA 1D Breps will be renumbered in

- **flag** (Flag)

Flag set on the IGA 1D Breps that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the IGA 1D Breps in model m flagged with f, from 1000000:

```
IGA1DBrep.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select IGA 1D Breps using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting IGA 1D Breps

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only IGA 1D Breps from that model can be selected. If the argument is a Flag then only IGA 1D Breps that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any IGA 1D Breps can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of IGA 1D Breps selected or null if menu cancelled

Return type

Number

Example

To select IGA 1D Breps from model m, flagging those selected with flag f, giving the prompt 'Select IGA 1D Breps':

```
IGA1DBrep.Select(f, 'Select IGA 1D Breps', m);
```

To select IGA 1D Breps, flagging those selected with flag f but limiting selection to IGA 1D Breps flagged with flag l, giving the prompt 'Select IGA 1D Breps':

```
IGA1DBrep.Select(f, 'Select IGA 1D Breps', l);
```

SetData(index*[integer]*, eid*[integer]*)

Description

Sets eid in *IGA_1D_BREP.

Arguments

- **index** (integer)

Index you want to set the eid for. **Note that indices start at 0**.

- **eid** (integer)

The ID of parametric edge.

Returns

No return value.

Example

To add an entry to the *IGA_1D_BREP ig1b with id 10 by using "entries" property to get the next available index:

```
var i = ig1b.entries;
ig1b.SetData(i, 10);
```

To change an entry in 3rd field for *IGA_1D_BREP ig1b with id 30:

```
ig1b.SetData(2, 30);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the IGA 1D Brep.

Arguments

- **flag** (Flag)

Flag to set on the IGA 1D Brep

Returns

No return value

Example

To set flag f for IGA 1D Brep ig1b:

```
ig1b.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the IGA 1D Brep. The IGA 1D Brep will be sketched until you either call IGA1DBrep.Unsketch(), IGA1DBrep.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 1D Brep is sketched. If omitted redraw is true. If you want to sketch several IGA 1D Breps and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch IGA 1D Brep ig1b:

```
ig1b.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged IGA 1D Breps in the model. The IGA 1D Breps will be sketched until you either call IGA1DBrep.Unsketch(), IGA1DBrep.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged IGA 1D Breps will be sketched in

- **flag** (Flag)

Flag set on the IGA 1D Breps that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 1D Breps are sketched. If omitted redraw is true. If you want to sketch flagged IGA 1D Breps several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all IGA 1D Breps flagged with flag in model m:

```
IGA1DBrep.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of IGA 1D Breps in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing IGA 1D Breps should be counted. If false or omitted referenced but undefined IGA 1D Breps will also be included in the total.

Returns

number of IGA 1D Breps

Return type

Number

Example

To get the total number of IGA 1D Breps in model m:

```
var total = IGA1DBrep.Total(m);
```

Unblank()

Description

Unblanks the IGA 1D Brep

Arguments

No arguments

Returns

No return value

Example

To unblank IGA 1D Brep ig1b:

```
ig1b.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the IGA 1D Breps in the model.

Arguments

- **Model** (Model)

Model that all IGA 1D Breps will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the IGA 1D Breps in model m:

```
IGA1DBrep.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged IGA 1D Breps in the model.

Arguments

- **Model** (Model)

Model that the flagged IGA 1D Breps will be unblanked in

- **flag** (Flag)

Flag set on the IGA 1D Breps that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the IGA 1D Breps in model m flagged with f:

```
IGA1DBrep.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the IGA 1D Breps in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all IGA 1D Breps will be unset in

- **flag** (Flag)

Flag to unset on the IGA 1D Breps

Returns

No return value

Example

To unset the flag f on all the IGA 1D Breps in model m:

```
IGA1DBrep.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the IGA 1D Brep.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 1D Brep is unsketched. If omitted redraw is true. If you want to unsketch several IGA 1D Breps and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch IGA 1D Brep ig1b:

```
ig1b.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all IGA 1D Breps.

Arguments

- **Model** (Model)

Model that all IGA 1D Breps will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 1D Breps are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all IGA 1D Breps in model m:

```
IGA1DBrep.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged IGA 1D Breps in the model.

Arguments

- **Model** (Model)

Model that all IGA 1D Breps will be unsketched in

- **flag** (Flag)

Flag set on the IGA 1D Breps that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 1D Breps are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all IGA 1D Breps flagged with flag in model m:

```
IGA1DBrep.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

IGA1DBrep object.

Return type

IGA1DBrep

Example

To check if IGA1DBrep property ig1b.example is a parameter by using the IGA1DBrep.GetParameter() method:

```
if (ig1b.ViewParameters().GetParameter(ig1b.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for IGA 1D Brep with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA 1D Brep object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for IGA 1D Brep ig1b:

```
ig1b.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for IGA 1D Brep ig1b:

```
ig1b.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for IGA 1D Brep. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for IGA 1D Brep ig1b:

```
ig1b.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this IGA 1D Brep.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for IGA 1D Brep ig1b:

```
var xrefs = ig1b.Xrefs();
```

---

toString()

Description

Creates a string containing the IGA 1d brep data in keyword format. Note that this contains the keyword header and the keyword cards. See also IGA1DBrep.Keyword() and IGA1DBrep.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for IGA 1d brep ig1b in keyword format

```
var s = ig1b.toString();
```

IGA1DNurbsUVW class

The IGA1DNurbsUVW class gives you access to IGA 1d nurbs uvw cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetControlPoint(index*[integer]*)
- GetKnotVector(index*[integer]*)
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetControlPoint(index*[integer]*, data*[Array of data]*)
- SetFlag(flag*[Flag]*)
- SetKnotVector(index*[integer]*, r*[real]*)
- Sketch(redraw (optional)*[boolean]*)
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

IGA1DNurbsUVW properties

Name	Type	Description
exists (read only)	logical	true if IGA 1D Nurbs UVW exists, false if referred to but not defined.
include	integer	The Include file number that the IGA 1D Nurbs UVW is in.
label	integer	IGA1DNurbsUVW number. Also see the patchid property which is an alternative name for this.
model (read only)	integer	The Model number that the IGA 1D Nurbs UVW is in.
nr	integer	Number of control points in the local r-direction.
patchid	integer	IGA1DNurbsUVW number. Also see the label property which is an alternative name for this.
pr	integer	Polynomial degree of the basis in the local r-direction.
rfirst	real	First knot value in the local r-direction.
rlast	real	Last knot value in the local r-direction.
unir	integer	Knot vector type in the local r-direction.

Detailed Description

The IGA1DNurbsUVW class allows you to create, modify, edit and manipulate IGA 1d nurbs uvw cards. See the documentation below for more details.

Constructor

new IGA1DNurbsUVW(Model*[Model]*, details*[object]*)

Description

Create a new IGA1DNurbsUVW object.

Arguments

- **Model** (Model)

Model that IGA 1d nurbs uvw will be created in

- **details** (object)

Details for creating the IGA1DNurbsUVW

Object has the following properties:

Name	Type	Description
nr (optional)	integer	Number of control points in the local r-direction.
patchid	integer	ID of the IGA 1D Nurbs UVW.
pr (optional)	integer	Polynomial degree of the basis in the local r-direction.
unir (optional)	integer	Knot vector type in the local r-direction.

Returns

IGA1DNurbsUVW object

Return type

IGA1DNurbsUVW

Example

To create a new IGA 1d nurbs uvw in model m of id 10:

```
var ig1u = new IGA1DNurbsUVW(m, {patchid:10});
```

To create a new IGA 1d nurbs uvw in model m of id 10 with nr, pr, unir as 2, 3, 4:

```
var ig1u = new IGA1DNurbsUVW(m, {patchid:10, nr:2, pr:3, unir:4});
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a IGA 1D Nurbs UVW.

Arguments

- **Comment** (Comment)

Comment that will be attached to the IGA 1D Nurbs UVW

Returns

No return value

Example

To associate comment c to the IGA 1D Nurbs UVW ig1u:

```
ig1u.AssociateComment(c);
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse IGA 1D Nurbs UVW ig1u:

```
ig1u.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the IGA 1D Nurbs UVW.

Arguments

- **flag** (Flag)

Flag to clear on the IGA 1D Nurbs UVW

Returns

No return value

Example

To clear flag f for IGA 1D Nurbs UVW ig1u:

```
ig1u.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the IGA 1D Nurbs UVW. The target include of the copied IGA 1D Nurbs UVW can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

IGA1DNurbsUVW object

Return type

IGA1DNurbsUVW

Example

To copy IGA 1D Nurbs UVW ig1u into IGA 1D Nurbs UVW z:

```
var z = ig1u.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a IGA 1D Nurbs UVW

Arguments

- **Model** (Model)

Model that the IGA 1D Nurbs UVW will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

IGA1DNurbsUVW object (or null if not made).

Return type

IGA1DNurbsUVW

Example

To start creating a IGA 1D Nurbs UVW in model m:

```
var ig1u = IGA1DNurbsUVW.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a IGA 1D Nurbs UVW.

Arguments

- **Comment** (Comment)

Comment that will be detached from the IGA 1D Nurbs UVW

Returns

No return value

Example

To detach comment c from the IGA 1D Nurbs UVW ig1u:

```
ig1u.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit IGA 1D Nurbs UVW ig1u:

```
ig1u.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for IGA 1D Nurbs UVW with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA 1D Nurbs UVW object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for IGA 1D Nurbs UVW ig1u:

```
ig1u.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for IGA 1D Nurbs UVW ig1u:

```
ig1u.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for IGA 1D Nurbs UVW. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for IGA 1D Nurbs UVW ig1u:

```
ig1u.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first IGA 1D Nurbs UVW in the model.

Arguments

- **Model** (Model)

Model to get first IGA 1D Nurbs UVW in

Returns

IGA1DNurbsUVW object (or null if there are no IGA 1D Nurbs UVWs in the model).

Return type

IGA1DNurbsUVW

Example

To get the first IGA 1D Nurbs UVW in model m:

```
var ig1u = IGA1DNurbsUVW.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free IGA 1D Nurbs UVW label in the model. Also see IGA1DNurbsUVW.LastFreeLabel(), IGA1DNurbsUVW.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free IGA 1D Nurbs UVW label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

IGA1DNurbsUVW label.

Return type

Number

Example

To get the first free IGA 1D Nurbs UVW label in model m:

```
var label = IGA1DNurbsUVW.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the IGA 1D Nurbs UVWs in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all IGA 1D Nurbs UVWs will be flagged in

- **flag** (Flag)

Flag to set on the IGA 1D Nurbs UVWs

Returns

No return value

Example

To flag all of the IGA 1D Nurbs UVWs with flag f in model m:

```
IGA1DNurbsUVW.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the IGA 1D Nurbs UVW is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the IGA 1D Nurbs UVW

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if IGA 1D Nurbs UVW ig1u has flag f set on it:

```
if (ig1u.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each IGA 1D Nurbs UVW in the model.
Note that ForEach has been designed to make looping over IGA 1D Nurbs UVWs as fast as possible and so has some limitations.
Firstly, a single temporary IGA1DNurbsUVW object is created and on each function call it is updated with the current IGA 1D Nurbs UVW data. This means that you should not try to store the IGA1DNurbsUVW object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new IGA 1D Nurbs UVWs inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all IGA 1D Nurbs UVWs are in

- **func** (function)

Function to call for each IGA 1D Nurbs UVW

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the IGA 1D Nurbs UVWs in model m:

```
IGA1DNurbsUVW.ForEach(m, test);
function test(ig1u)
{
// ig1u is IGA1DNurbsUVW object
}
```

To call function test for all of the IGA 1D Nurbs UVWs in model m with optional object:

```
var data = { x:0, y:0 };
IGA1DNurbsUVW.ForEach(m, test, data);
function test(ig1u, extra)
{
// ig1u is IGA1DNurbsUVW object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of IGA1DNurbsUVW objects or properties for all of the IGA 1D Nurbs UVWs in a model in PRIMER. If the optional property argument is not given then an array of IGA1DNurbsUVW objects is returned. If the property argument is given, that property value for each IGA 1D Nurbs UVW is returned in the array instead of a IGA1DNurbsUVW object

Arguments

- **Model** (Model)

Model to get IGA 1D Nurbs UVWs from

- **property (optional)** (string)

Name for property to get for all IGA 1D Nurbs UVWs in the model

Returns

Array of IGA1DNurbsUVW objects or properties

Return type

Array

Example

To make an array of IGA1DNurbsUVW objects for all of the IGA 1D Nurbs UVWs in model m:

```
var a = IGA1DNurbsUVW.GetAll(m);
```

To return an array containing the value of property ’foo’ (for example ’x’ for a node) for each IGA 1D Nurbs UVW in model m:

```
var a = IGA1DNurbsUVW.GetAll(m, ’foo’);
```

GetComments()

Description

Extracts the comments associated to a IGA 1D Nurbs UVW.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the IGA 1D Nurbs UVW ig1u:

```
var comm_array = ig1u.GetComments();
```

GetControlPoint(index*[integer]*)

Description

Returns the data for a specific control point an an array. There are nr control points.

Arguments

- **index** (integer)

Index you want the control data for. **Note that indices start at 0**.

Returns

An array containing the control point data.

Return type

Array

Example

To get the data for the 3rd control point in IGA 1d nurbs uvw ig1u:

```
var data = ig1u.GetControlPoint(2);
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of IGA1DNurbsUVW objects for all of the flagged IGA 1D Nurbs UVWs in a model in PRIMER If the optional property argument is not given then an array of IGA1DNurbsUVW objects is returned. If the property argument is given, then that property value for each IGA 1D Nurbs UVW is returned in the array instead of a IGA1DNurbsUVW object

Arguments

- **Model** (Model)

Model to get IGA 1D Nurbs UVWs from

- **flag** (Flag)

Flag set on the IGA 1D Nurbs UVWs that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged IGA 1D Nurbs UVWs in the model

Returns

Array of IGA1DNurbsUVW objects or properties

Return type

Array

Example

To make an array of IGA1DNurbsUVW objects for all of the IGA 1D Nurbs UVWs in model m flagged with f:

```
var ig1u = IGA1DNurbsUVW.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the IGA 1D Nurbs UVWs in model m flagged with f:

```
var a = IGA1DNurbsUVW.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the IGA1DNurbsUVW object for a IGA 1D Nurbs UVW ID.

Arguments

- **Model** (Model)

Model to find the IGA 1D Nurbs UVW in

- **number** (integer)

number of the IGA 1D Nurbs UVW you want the IGA1DNurbsUVW object for

Returns

IGA1DNurbsUVW object (or null if IGA 1D Nurbs UVW does not exist).

Return type

IGA1DNurbsUVW

Example

To get the IGA1DNurbsUVW object for IGA 1D Nurbs UVW 100 in model m

```
var ig1u = IGA1DNurbsUVW.GetFromID(m, 100);
```

GetKnotVector(index*[integer]*)

Description

The data for Knot values in the local r-direction.

Arguments

- **index** (integer)

Index you want the knot value for. **Note that indices start at 0**.

Returns

Knot values in the local r-direction.

Return type

Number

Example

To get the 3rd knot value in IGA 1d nurbs uvw ig1u:

```
var data = ig1u.GetKnotVector(2);
```

GetParameter(prop*[string]*)

Description

Checks if a IGA1DNurbsUVW property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the IGA1DNurbsUVW.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

IGA 1D Nurbs UVW property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if IGA1DNurbsUVW property ig1u.example is a parameter:

```
Options.property_parameter_names = true;
if (ig1u.GetParameter(ig1u.example) ) do_something...
Options.property_parameter_names = false;
```

To check if IGA1DNurbsUVW property ig1u.example is a parameter by using the GetParameter method:

```
if (ig1u.ViewParameters().GetParameter(ig1u.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this IGA 1d nurbs uvw (*IGA_1D_NURBS_UVW). **Note that a carriage return is not added**. See also IGA1DNurbsUVW.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for IGA 1d nurbs uvw ig1u:

```
var key = ig1u.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the IGA 1d nurbs uvw. **Note that a carriage return is not added**. See also IGA1DNurbsUVW.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for IGA 1d nurbs uvw ig1u:

```
var cards = ig1u.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last IGA 1D Nurbs UVW in the model.

Arguments

- **Model** (Model)

Model to get last IGA 1D Nurbs UVW in

Returns

IGA1DNurbsUVW object (or null if there are no IGA 1D Nurbs UVWs in the model).

Return type

IGA1DNurbsUVW

Example

To get the last IGA 1D Nurbs UVW in model m:

```
var ig1u = IGA1DNurbsUVW.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free IGA 1D Nurbs UVW label in the model. Also see IGA1DNurbsUVW.FirstFreeLabel(), IGA1DNurbsUVW.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free IGA 1D Nurbs UVW label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

IGA1DNurbsUVW label.

Return type

Number

Example

To get the last free IGA 1D Nurbs UVW label in model m:

```
var label = IGA1DNurbsUVW.LastFreeLabel(m);
```

Next()

Description

Returns the next IGA 1D Nurbs UVW in the model.

Arguments

No arguments

Returns

IGA1DNurbsUVW object (or null if there are no more IGA 1D Nurbs UVWs in the model).

Return type

IGA1DNurbsUVW

Example

To get the IGA 1D Nurbs UVW in model m after IGA 1D Nurbs UVW ig1u:

```
var ig1u = ig1u.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) IGA 1D Nurbs UVW label in the model. Also see IGA1DNurbsUVW.FirstFreeLabel(), IGA1DNurbsUVW.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free IGA 1D Nurbs UVW label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

IGA1DNurbsUVW label.

Return type

Number

Example

To get the next free IGA 1D Nurbs UVW label in model m:

```
var label = IGA1DNurbsUVW.NextFreeLabel(m);
```

Previous()

Description

Returns the previous IGA 1D Nurbs UVW in the model.

Arguments

No arguments

Returns

IGA1DNurbsUVW object (or null if there are no more IGA 1D Nurbs UVWs in the model).

Return type

IGA1DNurbsUVW

Example

To get the IGA 1D Nurbs UVW in model m before IGA 1D Nurbs UVW ig1u:

```
var ig1u = ig1u.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the IGA 1D Nurbs UVWs in the model.

Arguments

- **Model** (Model)

Model that all IGA 1D Nurbs UVWs will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the IGA 1D Nurbs UVWs in model m, from 1000000:

```
IGA1DNurbsUVW.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged IGA 1D Nurbs UVWs in the model.

Arguments

- **Model** (Model)

Model that all the flagged IGA 1D Nurbs UVWs will be renumbered in

- **flag** (Flag)

Flag set on the IGA 1D Nurbs UVWs that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the IGA 1D Nurbs UVWs in model m flagged with f, from 1000000:

```
IGA1DNurbsUVW.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select IGA 1D Nurbs UVWs using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting IGA 1D Nurbs UVWs

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only IGA 1D Nurbs UVWs from that model can be selected. If the argument is a Flag then only IGA 1D Nurbs UVWs that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any IGA 1D Nurbs UVWs can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of IGA 1D Nurbs UVWs selected or null if menu cancelled

Return type

Number

Example

To select IGA 1D Nurbs UVWs from model m, flagging those selected with flag f, giving the prompt 'Select IGA 1D Nurbs UVWs':

```
IGA1DNurbsUVW.Select(f, 'Select IGA 1D Nurbs UVWs', m);
```

To select IGA 1D Nurbs UVWs, flagging those selected with flag f but limiting selection to IGA 1D Nurbs UVWs flagged with flag l, giving the prompt 'Select IGA 1D Nurbs UVWs':

```
IGA1DNurbsUVW.Select(f, 'Select IGA 1D Nurbs UVWs', l);
```

SetControlPoint(index*[integer]*, data*[Array of data]*)

Description

Set the data for a specific control point. For each control point there will be control poin U, V, W and control weight WGT. There are nr control points.

Arguments

- **index** (integer)

Index you want the control point data for. **Note that indices start at 0**.

- **data** (Array of data)

Array containing the control point data. The array length should be 4.

Returns

No return value.

Example

To set the 3rd control point data for IGA 1d nurbs uvw ig1u to the values in array data:

```
ig1u.SetControlPoint(2, data);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the IGA 1D Nurbs UVW.

Arguments

- **flag** (Flag)

Flag to set on the IGA 1D Nurbs UVW

Returns

No return value

Example

To set flag f for IGA 1D Nurbs UVW ig1u:

```
ig1u.SetFlag(f);
```

SetKnotVector(index*[integer]*, r*[real]*)

Description

Sets knot value, r in *IGA_1D_NURBS_UVW. There are [nr + pr + 1] knot values.

Arguments

- **index** (integer)

Index you want to set the r for. **Note that indices start at 0**.

- **r** (real)

Knot values in the local r-direction

Returns

No return value.

Example

To set the 3rd knot value for IGA 1d nurbs uvw ig1u as 13.5:

```
ig1u.SetKnotVector(2, 13.5);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the IGA 1D Nurbs UVW. The IGA 1D Nurbs UVW will be sketched until you either call IGA1DNurbsUVW.Unsketch(), IGA1DNurbsUVW.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 1D Nurbs UVW is sketched. If omitted redraw is true. If you want to sketch several IGA 1D Nurbs UVWs and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch IGA 1D Nurbs UVW ig1u:

```
ig1u.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged IGA 1D Nurbs UVWs in the model. The IGA 1D Nurbs UVWs will be sketched until you either call IGA1DNurbsUVW.Unsketch(), IGA1DNurbsUVW.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged IGA 1D Nurbs UVWs will be sketched in

- **flag** (Flag)

Flag set on the IGA 1D Nurbs UVWs that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 1D Nurbs UVWs are sketched. If omitted redraw is true. If you want to sketch flagged IGA 1D Nurbs UVWs several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all IGA 1D Nurbs UVWs flagged with flag in model m:

```
IGA1DNurbsUVW.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of IGA 1D Nurbs UVWs in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing IGA 1D Nurbs UVWs should be counted. If false or omitted referenced but undefined IGA 1D Nurbs UVWs will also be included in the total.

Returns

number of IGA 1D Nurbs UVWs

Return type

Number

Example

To get the total number of IGA 1D Nurbs UVWs in model m:

```
var total = IGA1DNurbsUVW.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the IGA 1D Nurbs UVWs in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all IGA 1D Nurbs UVWs will be unset in

- **flag** (Flag)

Flag to unset on the IGA 1D Nurbs UVWs

Returns

No return value

Example

To unset the flag f on all the IGA 1D Nurbs UVWs in model m:

```
IGA1DNurbsUVW.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the IGA 1D Nurbs UVW.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 1D Nurbs UVW is unsketched. If omitted redraw is true. If you want to unsketch several IGA 1D Nurbs UVWs and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch IGA 1D Nurbs UVW ig1u:

```
ig1u.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all IGA 1D Nurbs UVWs.

Arguments

- **Model** (Model)

Model that all IGA 1D Nurbs UVWs will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 1D Nurbs UVWs are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all IGA 1D Nurbs UVWs in model m:

```
IGA1DNurbsUVW.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged IGA 1D Nurbs UVWs in the model.

Arguments

- **Model** (Model)

Model that all IGA 1D Nurbs UVWs will be unsketched in

- **flag** (Flag)

Flag set on the IGA 1D Nurbs UVWs that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 1D Nurbs UVWs are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all IGA 1D Nurbs UVWs flagged with flag in model m:

```
IGA1DNurbsUVW.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

IGA1DNurbsUVW object.

Return type

IGA1DNurbsUVW

Example

To check if IGA1DNurbsUVW property ig1u.example is a parameter by using the IGA1DNurbsUVW.GetParameter() method:

```
if (ig1u.ViewParameters().GetParameter(ig1u.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for IGA 1D Nurbs UVW with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA 1D Nurbs UVW object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for IGA 1D Nurbs UVW ig1u:

```
ig1u.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for IGA 1D Nurbs UVW ig1u:

```
ig1u.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for IGA 1D Nurbs UVW. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for IGA 1D Nurbs UVW ig1u:

```
ig1u.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this IGA 1D Nurbs UVW.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for IGA 1D Nurbs UVW ig1u:

```
var xrefs = ig1u.Xrefs();
```

---

toString()

Description

Creates a string containing the IGA 1d nurbs uvw data in keyword format. Note that this contains the keyword header and the keyword cards. See also IGA1DNurbsUVW.Keyword() and IGA1DNurbsUVW.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for IGA 1d nurbs uvw ig1u in keyword format

```
var s = ig1u.toString();
```

IGA1DNurbsXYZ class

The IGA1DNurbsXYZ class gives you access to IGA 1d nurbs xyz cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetControlPoint(index*[integer]*)
- GetKnotVector(index*[integer]*)
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetControlPoint(index*[integer]*, data*[Array of data]*)
- SetFlag(flag*[Flag]*)
- SetKnotVector(index*[integer]*, r*[real]*)
- Sketch(redraw (optional)*[boolean]*)
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

IGA1DNurbsXYZ properties

Name	Type	Description
exists (read only)	logical	true if IGA 1D Nurbs XYZ exists, false if referred to but not defined.
include	integer	The Include file number that the IGA 1D Nurbs XYZ is in.
label	integer	IGA1DNurbsXYZ number. Also see the patchid property which is an alternative name for this.
model (read only)	integer	The Model number that the IGA 1D Nurbs XYZ is in.
nr	integer	Number of control points in the local r-direction.
patchid	integer	IGA1DNurbsXYZ number. Also see the label property which is an alternative name for this.
pr	integer	Polynomial degree of the basis in the local r-direction.
rfirst	real	First knot value in the local r-direction.
rlast	real	Last knot value in the local r-direction.
unir	integer	Knot vector type in the local r-direction.

Detailed Description

The IGA1DNurbsXYZ class allows you to create, modify, edit and manipulate IGA 1d nurbs xyz cards. See the documentation below for more details.

Constructor

new IGA1DNurbsXYZ(Model*[Model]*, details*[object]*)

Description

Create a new IGA1DNurbsXYZ object.

Arguments

- **Model** (Model)

Model that IGA 1d nurbs xyz will be created in

- **details** (object)

Details for creating the IGA1DNurbsXYZ

Object has the following properties:

Name	Type	Description
nr (optional)	integer	Number of control points in the local r-direction.
patchid	integer	ID of the IGA 1D Nurbs XYZ.
pr (optional)	integer	Polynomial degree of the basis in the local r-direction.
unir (optional)	integer	Knot vector type in the local r-direction.

Returns

IGA1DNurbsXYZ object

Return type

IGA1DNurbsXYZ

Example

To create a new IGA 1d nurbs xyz in model m of id 10:

```
var ig1x = new IGA1DNurbsXYZ(m, {patchid:10});
```

To create a new IGA 1d nurbs xyz in model m of id 10 with nr, pr, unir as 2, 3, 4:

```
var ig1x = new IGA1DNurbsXYZ(m, {patchid:10, nr:2, pr:3, unir:4});
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a IGA 1D Nurbs XYZ.

Arguments

- **Comment** (Comment)

Comment that will be attached to the IGA 1D Nurbs XYZ

Returns

No return value

Example

To associate comment c to the IGA 1D Nurbs XYZ ig1x:

```
ig1x.AssociateComment(c);
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse IGA 1D Nurbs XYZ ig1x:

```
ig1x.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the IGA 1D Nurbs XYZ.

Arguments

- **flag** (Flag)

Flag to clear on the IGA 1D Nurbs XYZ

Returns

No return value

Example

To clear flag f for IGA 1D Nurbs XYZ ig1x:

```
ig1x.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the IGA 1D Nurbs XYZ. The target include of the copied IGA 1D Nurbs XYZ can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

IGA1DNurbsXYZ object

Return type

IGA1DNurbsXYZ

Example

To copy IGA 1D Nurbs XYZ ig1x into IGA 1D Nurbs XYZ z:

```
var z = ig1x.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a IGA 1D Nurbs XYZ

Arguments

- **Model** (Model)

Model that the IGA 1D Nurbs XYZ will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

IGA1DNurbsXYZ object (or null if not made).

Return type

IGA1DNurbsXYZ

Example

To start creating a IGA 1D Nurbs XYZ in model m:

```
var ig1x = IGA1DNurbsXYZ.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a IGA 1D Nurbs XYZ.

Arguments

- **Comment** (Comment)

Comment that will be detached from the IGA 1D Nurbs XYZ

Returns

No return value

Example

To detach comment c from the IGA 1D Nurbs XYZ ig1x:

```
ig1x.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit IGA 1D Nurbs XYZ ig1x:

```
ig1x.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for IGA 1D Nurbs XYZ with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA 1D Nurbs XYZ object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for IGA 1D Nurbs XYZ ig1x:

```
ig1x.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for IGA 1D Nurbs XYZ ig1x:

```
ig1x.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for IGA 1D Nurbs XYZ. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for IGA 1D Nurbs XYZ ig1x:

```
ig1x.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first IGA 1D Nurbs XYZ in the model.

Arguments

- **Model** (Model)

Model to get first IGA 1D Nurbs XYZ in

Returns

IGA1DNurbsXYZ object (or null if there are no IGA 1D Nurbs XYZs in the model).

Return type

IGA1DNurbsXYZ

Example

To get the first IGA 1D Nurbs XYZ in model m:

```
var ig1x = IGA1DNurbsXYZ.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free IGA 1D Nurbs XYZ label in the model. Also see IGA1DNurbsXYZ.LastFreeLabel(), IGA1DNurbsXYZ.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free IGA 1D Nurbs XYZ label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

IGA1DNurbsXYZ label.

Return type

Number

Example

To get the first free IGA 1D Nurbs XYZ label in model m:

```
var label = IGA1DNurbsXYZ.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the IGA 1D Nurbs XYZs in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all IGA 1D Nurbs XYZs will be flagged in

- **flag** (Flag)

Flag to set on the IGA 1D Nurbs XYZs

Returns

No return value

Example

To flag all of the IGA 1D Nurbs XYZs with flag f in model m:

```
IGA1DNurbsXYZ.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the IGA 1D Nurbs XYZ is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the IGA 1D Nurbs XYZ

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if IGA 1D Nurbs XYZ ig1x has flag f set on it:

```
if (ig1x.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each IGA 1D Nurbs XYZ in the model.
Note that ForEach has been designed to make looping over IGA 1D Nurbs XYZs as fast as possible and so has some limitations.
Firstly, a single temporary IGA1DNurbsXYZ object is created and on each function call it is updated with the current IGA 1D Nurbs XYZ data. This means that you should not try to store the IGA1DNurbsXYZ object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new IGA 1D Nurbs XYZs inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all IGA 1D Nurbs XYZs are in

- **func** (function)

Function to call for each IGA 1D Nurbs XYZ

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the IGA 1D Nurbs XYZs in model m:

```
IGA1DNurbsXYZ.ForEach(m, test);
function test(ig1x)
{
// ig1x is IGA1DNurbsXYZ object
}
```

To call function test for all of the IGA 1D Nurbs XYZs in model m with optional object:

```
var data = { x:0, y:0 };
IGA1DNurbsXYZ.ForEach(m, test, data);
function test(ig1x, extra)
{
// ig1x is IGA1DNurbsXYZ object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of IGA1DNurbsXYZ objects or properties for all of the IGA 1D Nurbs XYZs in a model in PRIMER. If the optional property argument is not given then an array of IGA1DNurbsXYZ objects is returned. If the property argument is given, that property value for each IGA 1D Nurbs XYZ is returned in the array instead of a IGA1DNurbsXYZ object

Arguments

- **Model** (Model)

Model to get IGA 1D Nurbs XYZs from

- **property (optional)** (string)

Name for property to get for all IGA 1D Nurbs XYZs in the model

Returns

Array of IGA1DNurbsXYZ objects or properties

Return type

Array

Example

To make an array of IGA1DNurbsXYZ objects for all of the IGA 1D Nurbs XYZs in model m:

```
var a = IGA1DNurbsXYZ.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each IGA 1D Nurbs XYZ in model m:

```
var a = IGA1DNurbsXYZ.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a IGA 1D Nurbs XYZ.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the IGA 1D Nurbs XYZ ig1x:

```
var comm_array = ig1x.GetComments();
```

GetControlPoint(index*[integer]*)

Description

Returns the data for a specific control point an an array. There are nr control points.

Arguments

- **index** (integer)

Index you want the control data for. **Note that indices start at 0**.

Returns

An array containing the control point data.

Return type

Array

Example

To get the data for the 3rd control point in IGA 1d nurbs xyz ig1x:

```
var data = ig1x.GetControlPoint(2);
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of IGA1DNurbsXYZ objects for all of the flagged IGA 1D Nurbs XYZs in a model in PRIMER If the optional property argument is not given then an array of IGA1DNurbsXYZ objects is returned. If the property argument is given, then that property value for each IGA 1D Nurbs XYZ is returned in the array instead of a IGA1DNurbsXYZ object

Arguments

- **Model** (Model)

Model to get IGA 1D Nurbs XYZs from

- **flag** (Flag)

Flag set on the IGA 1D Nurbs XYZs that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged IGA 1D Nurbs XYZs in the model

Returns

Array of IGA1DNurbsXYZ objects or properties

Return type

Array

Example

To make an array of IGA1DNurbsXYZ objects for all of the IGA 1D Nurbs XYZs in model m flagged with f:

```
var ig1x = IGA1DNurbsXYZ.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the IGA 1D Nurbs XYZs in model m flagged with f:

```
var a = IGA1DNurbsXYZ.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the IGA1DNurbsXYZ object for a IGA 1D Nurbs XYZ ID.

Arguments

- **Model** (Model)

Model to find the IGA 1D Nurbs XYZ in

- **number** (integer)

number of the IGA 1D Nurbs XYZ you want the IGA1DNurbsXYZ object for

Returns

IGA1DNurbsXYZ object (or null if IGA 1D Nurbs XYZ does not exist).

Return type

IGA1DNurbsXYZ

Example

To get the IGA1DNurbsXYZ object for IGA 1D Nurbs XYZ 100 in model m

```
var ig1x = IGA1DNurbsXYZ.GetFromID(m, 100);
```

GetKnotVector(index*[integer]*)

Description

The data for Knot values in the local r-direction.

Arguments

- **index** (integer)

Index you want the knot value for. **Note that indices start at 0**.

Returns

Knot values in the local r-direction.

Return type

Number

Example

To get the 3rd knot value in IGA 1d nurbs xyz ig1x:

```
var data = ig1x.GetKnotVector(2);
```

GetParameter(prop*[string]*)

Description

Checks if a IGA1DNurbsXYZ property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the IGA1DNurbsXYZ.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

IGA 1D Nurbs XYZ property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if IGA1DNurbsXYZ property ig1x.example is a parameter:

```
Options.property_parameter_names = true;
if (ig1x.GetParameter(ig1x.example) ) do_something...
Options.property_parameter_names = false;
```

To check if IGA1DNurbsXYZ property ig1x.example is a parameter by using the GetParameter method:

```
if (ig1x.ViewParameters().GetParameter(ig1x.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this IGA 1d nurbs xyz (*IGA_1D_NURBS_XYZ). **Note that a carriage return is not added**. See also IGA1DNurbsXYZ.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for IGA 1d nurbs xyz ig1x:

```
var key = ig1x.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the IGA 1d nurbs xyz. **Note that a carriage return is not added**. See also IGA1DNurbsXYZ.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for IGA 1d nurbs xyz ig1x:

```
var cards = ig1x.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last IGA 1D Nurbs XYZ in the model.

Arguments

- **Model** (Model)

Model to get last IGA 1D Nurbs XYZ in

Returns

IGA1DNurbsXYZ object (or null if there are no IGA 1D Nurbs XYZs in the model).

Return type

IGA1DNurbsXYZ

Example

To get the last IGA 1D Nurbs XYZ in model m:

```
var ig1x = IGA1DNurbsXYZ.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free IGA 1D Nurbs XYZ label in the model. Also see IGA1DNurbsXYZ.FirstFreeLabel(), IGA1DNurbsXYZ.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free IGA 1D Nurbs XYZ label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

IGA1DNurbsXYZ label.

Return type

Number

Example

To get the last free IGA 1D Nurbs XYZ label in model m:

```
var label = IGA1DNurbsXYZ.LastFreeLabel(m);
```

Next()

Description

Returns the next IGA 1D Nurbs XYZ in the model.

Arguments

No arguments

Returns

IGA1DNurbsXYZ object (or null if there are no more IGA 1D Nurbs XYZs in the model).

Return type

IGA1DNurbsXYZ

Example

To get the IGA 1D Nurbs XYZ in model m after IGA 1D Nurbs XYZ ig1x:

```
var ig1x = ig1x.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) IGA 1D Nurbs XYZ label in the model. Also see IGA1DNurbsXYZ.FirstFreeLabel(), IGA1DNurbsXYZ.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free IGA 1D Nurbs XYZ label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

IGA1DNurbsXYZ label.

Return type

Number

Example

To get the next free IGA 1D Nurbs XYZ label in model m:

```
var label = IGA1DNurbsXYZ.NextFreeLabel(m);
```

Previous()

Description

Returns the previous IGA 1D Nurbs XYZ in the model.

Arguments

No arguments

Returns

IGA1DNurbsXYZ object (or null if there are no more IGA 1D Nurbs XYZs in the model).

Return type

IGA1DNurbsXYZ

Example

To get the IGA 1D Nurbs XYZ in model m before IGA 1D Nurbs XYZ ig1x:

```
var ig1x = ig1x.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the IGA 1D Nurbs XYZs in the model.

Arguments

- **Model** (Model)

Model that all IGA 1D Nurbs XYZs will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the IGA 1D Nurbs XYZs in model m, from 1000000:

```
IGA1DNurbsXYZ.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged IGA 1D Nurbs XYZs in the model.

Arguments

- **Model** (Model)

Model that all the flagged IGA 1D Nurbs XYZs will be renumbered in

- **flag** (Flag)

Flag set on the IGA 1D Nurbs XYZs that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the IGA 1D Nurbs XYZs in model m flagged with f, from 1000000:

```
IGA1DNurbsXYZ.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select IGA 1D Nurbs XYZs using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting IGA 1D Nurbs XYZs

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only IGA 1D Nurbs XYZs from that model can be selected. If the argument is a Flag then only IGA 1D Nurbs XYZs that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any IGA 1D Nurbs XYZs can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of IGA 1D Nurbs XYZs selected or null if menu cancelled

Return type

Number

Example

To select IGA 1D Nurbs XYZs from model m, flagging those selected with flag f, giving the prompt 'Select IGA 1D Nurbs XYZs':

```
IGA1DNurbsXYZ.Select(f, 'Select IGA 1D Nurbs XYZs', m);
```

To select IGA 1D Nurbs XYZs, flagging those selected with flag f but limiting selection to IGA 1D Nurbs XYZs flagged with flag l, giving the prompt 'Select IGA 1D Nurbs XYZs':

```
IGA1DNurbsXYZ.Select(f, 'Select IGA 1D Nurbs XYZs', l);
```

SetControlPoint(index*[integer]*, data*[Array of data]*)

Description

Set the data for a specific control point. For each control point there will be control point X, Y, Z and control weight WGT. There are nr control points.

Arguments

- **index** (integer)

Index you want the control point data for. **Note that indices start at 0**.

- **data** (Array of data)

Array containing the control point data. The array length should be 4.

Returns

No return value.

Example

To set the 3rd control point data for IGA 1d nurbs xyz ig1x to the values in array data:

```
ig1x.SetControlPoint(2, data);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the IGA 1D Nurbs XYZ.

Arguments

- **flag** (Flag)

Flag to set on the IGA 1D Nurbs XYZ

Returns

No return value

Example

To set flag f for IGA 1D Nurbs XYZ ig1x:

```
ig1x.SetFlag(f);
```

SetKnotVector(index*[integer]*, r*[real]*)

Description

Sets knot value, r in *IGA_1D_NURBS_XYZ. There are [nr + pr + 1] knot values.

Arguments

- **index** (integer)

Index you want to set the r for. **Note that indices start at 0**.

- **r** (real)

Knot values in the local r-direction

Returns

No return value.

Example

To set the 3rd knot value for IGA 1d nurbs xyz ig1x as 13.5:

```
ig1x.SetKnotVector(2, 13.5);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the IGA 1D Nurbs XYZ. The IGA 1D Nurbs XYZ will be sketched until you either call IGA1DNurbsXYZ.Unsketch(), IGA1DNurbsXYZ.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 1D Nurbs XYZ is sketched. If omitted redraw is true. If you want to sketch several IGA 1D Nurbs XYZs and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch IGA 1D Nurbs XYZ ig1x:

```
ig1x.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged IGA 1D Nurbs XYZs in the model. The IGA 1D Nurbs XYZs will be sketched until you either call IGA1DNurbsXYZ.Unsketch(), IGA1DNurbsXYZ.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged IGA 1D Nurbs XYZs will be sketched in

- **flag** (Flag)

Flag set on the IGA 1D Nurbs XYZs that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 1D Nurbs XYZs are sketched. If omitted redraw is true. If you want to sketch flagged IGA 1D Nurbs XYZs several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all IGA 1D Nurbs XYZs flagged with flag in model m:

```
IGA1DNurbsXYZ.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of IGA 1D Nurbs XYZs in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing IGA 1D Nurbs XYZs should be counted. If false or omitted referenced but undefined IGA 1D Nurbs XYZs will also be included in the total.

Returns

number of IGA 1D Nurbs XYZs

Return type

Number

Example

To get the total number of IGA 1D Nurbs XYZs in model m:

```
var total = IGA1DNurbsXYZ.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the IGA 1D Nurbs XYZs in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all IGA 1D Nurbs XYZs will be unset in

- **flag** (Flag)

Flag to unset on the IGA 1D Nurbs XYZs

Returns

No return value

Example

To unset the flag f on all the IGA 1D Nurbs XYZs in model m:

```
IGA1DNurbsXYZ.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the IGA 1D Nurbs XYZ.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 1D Nurbs XYZ is unsketched. If omitted redraw is true. If you want to unsketch several IGA 1D Nurbs XYZs and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch IGA 1D Nurbs XYZ ig1x:

```
ig1x.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all IGA 1D Nurbs XYZs.

Arguments

- **Model** (Model)

Model that all IGA 1D Nurbs XYZs will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 1D Nurbs XYZs are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all IGA 1D Nurbs XYZs in model m:

```
IGA1DNurbsXYZ.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged IGA 1D Nurbs XYZs in the model.

Arguments

- **Model** (Model)

Model that all IGA 1D Nurbs XYZs will be unsketched in

- **flag** (Flag)

Flag set on the IGA 1D Nurbs XYZs that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 1D Nurbs XYZs are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all IGA 1D Nurbs XYZs flagged with flag in model m:

```
IGA1DNurbsXYZ.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

IGA1DNurbsXYZ object.

Return type

IGA1DNurbsXYZ

Example

To check if IGA1DNurbsXYZ property ig1x.example is a parameter by using the IGA1DNurbsXYZ.GetParameter() method:

```
if (ig1x.ViewParameters().GetParameter(ig1x.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for IGA 1D Nurbs XYZ with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA 1D Nurbs XYZ object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for IGA 1D Nurbs XYZ ig1x:

```
ig1x.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for IGA 1D Nurbs XYZ ig1x:

```
ig1x.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for IGA 1D Nurbs XYZ. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for IGA 1D Nurbs XYZ ig1x:

```
ig1x.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this IGA 1D Nurbs XYZ.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for IGA 1D Nurbs XYZ ig1x:

```
var xrefs = ig1x.Xrefs();
```

---

toString()

Description

Creates a string containing the IGA 1d nurbs xyz data in keyword format. Note that this contains the keyword header and the keyword cards. See also IGA1DNurbsXYZ.Keyword() and IGA1DNurbsXYZ.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for IGA 1d nurbs xyz ig1x in keyword format

```
var s = ig1x.toString();
```

IGA2DBasisTransformXYZ class

The IGA2DBasisTransformXYZ class gives you access to IGA 2d basis transform xyz cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

IGA2DBasisTransformXYZ properties

Name	Type	Description

exists (read only)	logical	true if IGA 2D Basis Transform XYZ exists, false if referred to but not defined.
filename	string	Number of control points in the local r-direction.
include	integer	The Include file number that the IGA 2D Basis Transform XYZ is in.
label	integer	IGA2DBasisTransformXYZ number. Also see the patchid property which is an alternative name for this.
model (read only)	integer	The Model number that the IGA 2D Basis Transform XYZ is in.
patchid	integer	IGA2DBasisTransformXYZ number. Also see the label property which is an alternative name for this.

Detailed Description

The IGA2DBasisTransformXYZ class allows you to create, modify, edit and manipulate IGA 2d basis transform xyz cards. See the documentation below for more details.

Constructor

new IGA2DBasisTransformXYZ(Model*[Model]*, details*[object]*)

Description

Create a new IGA2DBasisTransformXYZ object.

Arguments

- **Model** (Model)

Model that IGA 2d basis transform xyz will be created in

- **details** (object)

Details for creating the IGA2DBasisTransformXYZ

Object has the following properties:

Name	Type	Description
filename (optional)	string	Name of file containing patch data.
patchid	integer	ID of the IGA 2D Basis Transform XYZ.

Returns

IGA2DBasisTransformXYZ object

Return type

IGA2DBasisTransformXYZ

Example

To create a new IGA 2d basis transform xyz in model m of id 10:

```
var ig2bt = new IGA2DBasisTransformXYZ(m, {patchid:10});
```

To create a new IGA 2d basis transform xyz in model m of id 10 with a filename:

```
var ig2bt = new IGA2DBasisTransformXYZ(m, {patchid:10,
filename:"Filename\\with\\path.extension"});
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a IGA 2D Basis Transform XYZ.

Arguments

- **Comment** (Comment)

Comment that will be attached to the IGA 2D Basis Transform XYZ

Returns

No return value

Example

To associate comment c to the IGA 2D Basis Transform XYZ ig2bt:

```
ig2bt.AssociateComment(c);
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse IGA 2D Basis Transform XYZ ig2bt:

```
ig2bt.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the IGA 2D Basis Transform XYZ.

Arguments

- **flag** (Flag)

Flag to clear on the IGA 2D Basis Transform XYZ

Returns

No return value

Example

To clear flag f for IGA 2D Basis Transform XYZ ig2bt:

```
ig2bt.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the IGA 2D Basis Transform XYZ. The target include of the copied IGA 2D Basis Transform XYZ can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

IGA2DBasisTransformXYZ object

Return type

IGA2DBasisTransformXYZ

Example

To copy IGA 2D Basis Transform XYZ ig2bt into IGA 2D Basis Transform XYZ z:

```
var z = ig2bt.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a IGA 2D Basis Transform XYZ

Arguments

- **Model** (Model)

Model that the IGA 2D Basis Transform XYZ will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

IGA2DBasisTransformXYZ object (or null if not made).

Return type

IGA2DBasisTransformXYZ

Example

To start creating a IGA 2D Basis Transform XYZ in model m:

```
var ig2bt = IGA2DBasisTransformXYZ.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a IGA 2D Basis Transform XYZ.

Arguments

- **Comment** (Comment)

Comment that will be detached from the IGA 2D Basis Transform XYZ

Returns

No return value

Example

To detach comment c from the IGA 2D Basis Transform XYZ ig2bt:

```
ig2bt.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit IGA 2D Basis Transform XYZ ig2bt:

```
ig2bt.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for IGA 2D Basis Transform XYZ with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description

autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA 2D Basis Transform XYZ object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for IGA 2D Basis Transform XYZ ig2bt:

```
ig2bt.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for IGA 2D Basis Transform XYZ ig2bt:

```
ig2bt.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for IGA 2D Basis Transform XYZ. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for IGA 2D Basis Transform XYZ ig2bt:

```
ig2bt.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first IGA 2D Basis Transform XYZ in the model.

Arguments

- **Model** (Model)

Model to get first IGA 2D Basis Transform XYZ in

Returns

IGA2DBasisTransformXYZ object (or null if there are no IGA 2D Basis Transform XYZs in the model).

Return type

IGA2DBasisTransformXYZ

Example

To get the first IGA 2D Basis Transform XYZ in model m:

```
var ig2bt = IGA2DBasisTransformXYZ.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free IGA 2D Basis Transform XYZ label in the model. Also see IGA2DBasisTransformXYZ.LastFreeLabel(), IGA2DBasisTransformXYZ.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free IGA 2D Basis Transform XYZ label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

IGA2DBasisTransformXYZ label.

Return type

Number

Example

To get the first free IGA 2D Basis Transform XYZ label in model m:

```
var label = IGA2DBasisTransformXYZ.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the IGA 2D Basis Transform XYZs in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all IGA 2D Basis Transform XYZs will be flagged in

- **flag** (Flag)

Flag to set on the IGA 2D Basis Transform XYZs

Returns

No return value

Example

To flag all of the IGA 2D Basis Transform XYZs with flag f in model m:

```
IGA2DBasisTransformXYZ.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the IGA 2D Basis Transform XYZ is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the IGA 2D Basis Transform XYZ

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if IGA 2D Basis Transform XYZ ig2bt has flag f set on it:

```
if (ig2bt.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each IGA 2D Basis Transform XYZ in the model.
Note that ForEach has been designed to make looping over IGA 2D Basis Transform XYZs as fast as possible and so has some limitations.
Firstly, a single temporary IGA2DBasisTransformXYZ object is created and on each function call it is updated with the current IGA 2D Basis Transform XYZ data. This means that you should not try to store the IGA2DBasisTransformXYZ object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new IGA 2D Basis Transform XYZs inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all IGA 2D Basis Transform XYZs are in

- **func** (function)

Function to call for each IGA 2D Basis Transform XYZ

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the IGA 2D Basis Transform XYZs in model m:

```
IGA2DBasisTransformXYZ.ForEach(m, test);
function test(ig2bt)
{
// ig2bt is IGA2DBasisTransformXYZ object
}
```

To call function test for all of the IGA 2D Basis Transform XYZs in model m with optional object:

```
var data = { x:0, y:0 };
IGA2DBasisTransformXYZ.ForEach(m, test, data);
function test(ig2bt, extra)
{
// ig2bt is IGA2DBasisTransformXYZ object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of IGA2DBasisTransformXYZ objects or properties for all of the IGA 2D Basis Transform XYZs in a model in PRIMER. If the optional property argument is not given then an array of IGA2DBasisTransformXYZ objects is returned. If the property argument is given, that property value for each IGA 2D Basis Transform XYZ is returned in the array instead of a IGA2DBasisTransformXYZ object

Arguments

- **Model** (Model)

Model to get IGA 2D Basis Transform XYZs from

- **property (optional)** (string)

Name for property to get for all IGA 2D Basis Transform XYZs in the model

Returns

Array of IGA2DBasisTransformXYZ objects or properties

Return type

Array

Example

To make an array of IGA2DBasisTransformXYZ objects for all of the IGA 2D Basis Transform XYZs in model m:

```
var a = IGA2DBasisTransformXYZ.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each IGA 2D Basis Transform XYZ in model m:

```
var a = IGA2DBasisTransformXYZ.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a IGA 2D Basis Transform XYZ.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the IGA 2D Basis Transform XYZ ig2bt:

```
var comm_array = ig2bt.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of IGA2DBasisTransformXYZ objects for all of the flagged IGA 2D Basis Transform XYZs in a model in PRIMER If the optional property argument is not given then an array of IGA2DBasisTransformXYZ objects is returned. If the property argument is given, then that property value for each IGA 2D Basis Transform XYZ is returned in the array instead of a IGA2DBasisTransformXYZ object

Arguments

- **Model** (Model)

Model to get IGA 2D Basis Transform XYZs from

- **flag** (Flag)

Flag set on the IGA 2D Basis Transform XYZs that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged IGA 2D Basis Transform XYZs in the model

Returns

Array of IGA2DBasisTransformXYZ objects or properties

Return type

Array

Example

To make an array of IGA2DBasisTransformXYZ objects for all of the IGA 2D Basis Transform XYZs in model m flagged with f:

```
var ig2bt = IGA2DBasisTransformXYZ.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the IGA 2D Basis Transform XYZs in model m flagged with f:

```
var a = IGA2DBasisTransformXYZ.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the IGA2DBasisTransformXYZ object for a IGA 2D Basis Transform XYZ ID.

Arguments

- **Model** (Model)

Model to find the IGA 2D Basis Transform XYZ in

- **number** (integer)

number of the IGA 2D Basis Transform XYZ you want the IGA2DBasisTransformXYZ object for

Returns

IGA2DBasisTransformXYZ object (or null if IGA 2D Basis Transform XYZ does not exist).

Return type

IGA2DBasisTransformXYZ

Example

To get the IGA2DBasisTransformXYZ object for IGA 2D Basis Transform XYZ 100 in model m

```
var ig2bt = IGA2DBasisTransformXYZ.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a IGA2DBasisTransformXYZ property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the IGA2DBasisTransformXYZ.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

IGA 2D Basis Transform XYZ property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if IGA2DBasisTransformXYZ property ig2bt.example is a parameter:

```
Options.property_parameter_names = true;
if (ig2bt.GetParameter(ig2bt.example) ) do_something...
Options.property_parameter_names = false;
```

To check if IGA2DBasisTransformXYZ property ig2bt.example is a parameter by using the GetParameter method:

```
if (ig2bt.ViewParameters().GetParameter(ig2bt.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this IGA 2d basis transform xyz (*IGA_2D_BASIS_TRANSFORM_XYZ). **Note that a carriage return is not added**. See also IGA2DBasisTransformXYZ.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for IGA 2d basis transform xyz ig2bt:

```
var key = ig2bt.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the IGA 2d basis transform xyz. **Note that a carriage return is not added**. See also IGA2DBasisTransformXYZ.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for IGA 2d basis transform xyz ig2bt:

```
var cards = ig2bt.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last IGA 2D Basis Transform XYZ in the model.

Arguments

- **Model** (Model)

Model to get last IGA 2D Basis Transform XYZ in

Returns

IGA2DBasisTransformXYZ object (or null if there are no IGA 2D Basis Transform XYZs in the model).

Return type

IGA2DBasisTransformXYZ

Example

To get the last IGA 2D Basis Transform XYZ in model m:

```
var ig2bt = IGA2DBasisTransformXYZ.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free IGA 2D Basis Transform XYZ label in the model. Also see IGA2DBasisTransformXYZ.FirstFreeLabel(), IGA2DBasisTransformXYZ.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free IGA 2D Basis Transform XYZ label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

IGA2DBasisTransformXYZ label.

Return type

Number

Example

To get the last free IGA 2D Basis Transform XYZ label in model m:

```
var label = IGA2DBasisTransformXYZ.LastFreeLabel(m);
```

Next()

Description

Returns the next IGA 2D Basis Transform XYZ in the model.

Arguments

No arguments

Returns

IGA2DBasisTransformXYZ object (or null if there are no more IGA 2D Basis Transform XYZs in the model).

Return type

IGA2DBasisTransformXYZ

Example

To get the IGA 2D Basis Transform XYZ in model m after IGA 2D Basis Transform XYZ ig2bt:

```
var ig2bt = ig2bt.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) IGA 2D Basis Transform XYZ label in the model. Also see IGA2DBasisTransformXYZ.FirstFreeLabel(), IGA2DBasisTransformXYZ.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free IGA 2D Basis Transform XYZ label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

IGA2DBasisTransformXYZ label.

Return type

Number

Example

To get the next free IGA 2D Basis Transform XYZ label in model m:

```
var label = IGA2DBasisTransformXYZ.NextFreeLabel(m);
```

Previous()

Description

Returns the previous IGA 2D Basis Transform XYZ in the model.

Arguments

No arguments

Returns

IGA2DBasisTransformXYZ object (or null if there are no more IGA 2D Basis Transform XYZs in the model).

Return type

IGA2DBasisTransformXYZ

Example

To get the IGA 2D Basis Transform XYZ in model m before IGA 2D Basis Transform XYZ ig2bt:

```
var ig2bt = ig2bt.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the IGA 2D Basis Transform XYZs in the model.

Arguments

- **Model** (Model)

Model that all IGA 2D Basis Transform XYZs will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the IGA 2D Basis Transform XYZs in model m, from 1000000:

```
IGA2DBasisTransformXYZ.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged IGA 2D Basis Transform XYZs in the model.

Arguments

- **Model** (Model)

Model that all the flagged IGA 2D Basis Transform XYZs will be renumbered in

- **flag** (Flag)

Flag set on the IGA 2D Basis Transform XYZs that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the IGA 2D Basis Transform XYZs in model m flagged with f, from 1000000:

```
IGA2DBasisTransformXYZ.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select IGA 2D Basis Transform XYZs using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting IGA 2D Basis Transform XYZs

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only IGA 2D Basis Transform XYZs from that model can be selected. If the argument is a Flag then only IGA 2D Basis Transform XYZs that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any IGA 2D Basis Transform XYZs can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of IGA 2D Basis Transform XYZs selected or null if menu cancelled

Return type

Number

Example

To select IGA 2D Basis Transform XYZs from model m, flagging those selected with flag f, giving the prompt 'Select IGA 2D Basis Transform XYZs':

```
IGA2DBasisTransformXYZ.Select(f, 'Select IGA 2D Basis Transform XYZs', m);
```

To select IGA 2D Basis Transform XYZs, flagging those selected with flag f but limiting selection to IGA 2D Basis Transform XYZs flagged with flag l, giving the prompt 'Select IGA 2D Basis Transform XYZs':

```
IGA2DBasisTransformXYZ.Select(f, 'Select IGA 2D Basis Transform XYZs', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the IGA 2D Basis Transform XYZ.

Arguments

- **flag** (Flag)

Flag to set on the IGA 2D Basis Transform XYZ

Returns

No return value

Example

To set flag f for IGA 2D Basis Transform XYZ ig2bt:

```
ig2bt.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the IGA 2D Basis Transform XYZ. The IGA 2D Basis Transform XYZ will be sketched until you either call IGA2DBasisTransformXYZ.Unsketch(), IGA2DBasisTransformXYZ.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 2D Basis Transform XYZ is sketched. If omitted redraw is true. If you want to sketch several IGA 2D Basis Transform XYZs and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch IGA 2D Basis Transform XYZ ig2bt:

```
ig2bt.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged IGA 2D Basis Transform XYZs in the model. The IGA 2D Basis Transform XYZs will be sketched until you either call IGA2DBasisTransformXYZ.Unsketch(), IGA2DBasisTransformXYZ.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged IGA 2D Basis Transform XYZs will be sketched in

- **flag** (Flag)

Flag set on the IGA 2D Basis Transform XYZs that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 2D Basis Transform XYZs are sketched. If omitted redraw is true. If you want to sketch flagged IGA 2D Basis Transform XYZs several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all IGA 2D Basis Transform XYZs flagged with flag in model m:

```
IGA2DBasisTransformXYZ.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of IGA 2D Basis Transform XYZs in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing IGA 2D Basis Transform XYZs should be counted. If false or omitted referenced but undefined IGA 2D Basis Transform XYZs will also be included in the total.

Returns

number of IGA 2D Basis Transform XYZs

Return type

Number

Example

To get the total number of IGA 2D Basis Transform XYZs in model m:

```
var total = IGA2DBasisTransformXYZ.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the IGA 2D Basis Transform XYZs in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all IGA 2D Basis Transform XYZs will be unset in

- **flag** (Flag)

Flag to unset on the IGA 2D Basis Transform XYZs

Returns

No return value

Example

To unset the flag f on all the IGA 2D Basis Transform XYZs in model m:

```
IGA2DBasisTransformXYZ.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the IGA 2D Basis Transform XYZ.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 2D Basis Transform XYZ is unsketched. If omitted redraw is true. If you want to unsketch several IGA 2D Basis Transform XYZs and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch IGA 2D Basis Transform XYZ ig2bt:

```
ig2bt.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all IGA 2D Basis Transform XYZs.

Arguments

- **Model** (Model)

Model that all IGA 2D Basis Transform XYZs will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 2D Basis Transform XYZs are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all IGA 2D Basis Transform XYZs in model m:

```
IGA2DBasisTransformXYZ.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged IGA 2D Basis Transform XYZs in the model.

Arguments

- **Model** (Model)

Model that all IGA 2D Basis Transform XYZs will be unsketched in

- **flag** (Flag)

Flag set on the IGA 2D Basis Transform XYZs that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 2D Basis Transform XYZs are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all IGA 2D Basis Transform XYZs flagged with flag in model m:

```
IGA2DBasisTransformXYZ.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

IGA2DBasisTransformXYZ object.

Return type

IGA2DBasisTransformXYZ

Example

To check if IGA2DBasisTransformXYZ property ig2bt.example is a parameter by using the IGA2DBasisTransformXYZ.GetParameter() method:

```
if (ig2bt.ViewParameters().GetParameter(ig2bt.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for IGA 2D Basis Transform XYZ with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA 2D Basis Transform XYZ object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for IGA 2D Basis Transform XYZ ig2bt:

```
ig2bt.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for IGA 2D Basis Transform XYZ ig2bt:

```
ig2bt.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for IGA 2D Basis Transform XYZ. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for IGA 2D Basis Transform XYZ ig2bt:

```
ig2bt.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this IGA 2D Basis Transform XYZ.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for IGA 2D Basis Transform XYZ ig2bt:

```
var xrefs = ig2bt.Xrefs();
```

---

toString()

Description

Creates a string containing the IGA 2d basis transform xyz data in keyword format. Note that this contains the keyword header and the keyword cards. See also IGA2DBasisTransformXYZ.Keyword() and IGA2DBasisTransformXYZ.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for IGA 2d basis transform xyz ig2bt in keyword format

```
var s = ig2bt.toString();
```

IGA2DBrep class

The IGA2DBrep class gives you access to IGA 2d brep cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetData(index*[integer]*)
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- RemoveData(index*[integer]*)
- SetData(index*[integer]*, fid*[integer]*)
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

IGA2DBrep properties

Name	Type	Description
brid	integer	IGA2DBrep number. Also see the label property which is an alternative name for this.
entries (read only)	integer	Number of Parametric Face IDs (fid) added in IGA_2D_BREP. Can be used as index to add a new entry.
exists (read only)	logical	true if IGA 2D Brep exists, false if referred to but not defined.
include	integer	The Include file number that the IGA 2D Brep is in.
label	integer	IGA2DBrep number. Also see the brid property which is an alternative name for this.
model (read only)	integer	The Model number that the IGA 2D Brep is in.

Detailed Description

The IGA2DBrep class allows you to create, modify, edit and manipulate IGA 2d brep cards. See the documentation below for more details.

Constructor

new IGA2DBrep(Model*[Model]*, details*[object]*)

Description

Create a new IGA2DBrep object.

Arguments

- **Model** (Model)

Model that IGA 2d brep will be created in

- **details** (object)

Details for creating the IGA2DBrep

Object has the following properties:

Name	Type	Description
brid	integer	ID of the IGA 2D Brep.

Returns

IGA2DBrep object

Return type

IGA2DBrep

Example

To create a new IGA 2d brep in model m of id 10:

```
var ig2b = new IGA2DBrep(m, {brid:10});
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a IGA 2D Brep.

Arguments

- **Comment** (Comment)

Comment that will be attached to the IGA 2D Brep

Returns

No return value

Example

To associate comment c to the IGA 2D Brep ig2b:

```
ig2b.AssociateComment(c);
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse IGA 2D Brep ig2b:

```
ig2b.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the IGA 2D Brep.

Arguments

- **flag** (Flag)

Flag to clear on the IGA 2D Brep

Returns

No return value

Example

To clear flag f for IGA 2D Brep ig2b:

```
ig2b.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the IGA 2D Brep. The target include of the copied IGA 2D Brep can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

IGA2DBrep object

Return type

IGA2DBrep

Example

To copy IGA 2D Brep ig2b into IGA 2D Brep z:

```
var z = ig2b.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a IGA 2D Brep

Arguments

- **Model** (Model)

Model that the IGA 2D Brep will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

IGA2DBrep object (or null if not made).

Return type

IGA2DBrep

Example

To start creating a IGA 2D Brep in model m:

```
var ig2b = IGA2DBrep.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a IGA 2D Brep.

Arguments

- **Comment** (Comment)

Comment that will be detached from the IGA 2D Brep

Returns

No return value

Example

To detach comment c from the IGA 2D Brep ig2b:

```
ig2b.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit IGA 2D Brep ig2b:

```
ig2b.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for IGA 2D Brep with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA 2D Brep object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for IGA 2D Brep ig2b:

```
ig2b.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for IGA 2D Brep ig2b:

```
ig2b.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for IGA 2D Brep. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for IGA 2D Brep ig2b:

```
ig2b.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first IGA 2D Brep in the model.

Arguments

- **Model** (Model)

Model to get first IGA 2D Brep in

Returns

IGA2DBrep object (or null if there are no IGA 2D Breps in the model).

Return type

IGA2DBrep

Example

To get the first IGA 2D Brep in model m:

```
var ig2b = IGA2DBrep.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free IGA 2D Brep label in the model. Also see IGA2DBrep.LastFreeLabel(), IGA2DBrep.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free IGA 2D Brep label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

IGA2DBrep label.

Return type

Number

Example

To get the first free IGA 2D Brep label in model m:

```
var label = IGA2DBrep.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the IGA 2D Breps in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all IGA 2D Breps will be flagged in

- **flag** (Flag)

Flag to set on the IGA 2D Breps

Returns

No return value

Example

To flag all of the IGA 2D Breps with flag f in model m:

```
IGA2DBrep.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the IGA 2D Brep is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the IGA 2D Brep

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if IGA 2D Brep ig2b has flag f set on it:

```
if (ig2b.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each IGA 2D Brep in the model.
Note that ForEach has been designed to make looping over IGA 2D Breps as fast as possible and so has some limitations.
Firstly, a single temporary IGA2DBrep object is created and on each function call it is updated with the current IGA 2D Brep data. This means that you should not try to store the IGA2DBrep object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new IGA 2D Breps inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all IGA 2D Breps are in

- **func** (function)

Function to call for each IGA 2D Brep

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the IGA 2D Breps in model m:

```
IGA2DBrep.ForEach(m, test);
function test(ig2b)
{
// ig2b is IGA2DBrep object
}
```

To call function test for all of the IGA 2D Breps in model m with optional object:

```
var data = { x:0, y:0 };
IGA2DBrep.ForEach(m, test, data);
function test(ig2b, extra)
{
// ig2b is IGA2DBrep object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of IGA2DBrep objects or properties for all of the IGA 2D Breps in a model in PRIMER. If the optional property argument is not given then an array of IGA2DBrep objects is returned. If the property argument is given, that property value for each IGA 2D Brep is returned in the array instead of a IGA2DBrep object

Arguments

- **Model** (Model)

Model to get IGA 2D Breps from

- **property (optional)** (string)

Name for property to get for all IGA 2D Breps in the model

Returns

Array of IGA2DBrep objects or properties

Return type

Array

Example

To make an array of IGA2DBrep objects for all of the IGA 2D Breps in model m:

```
var a = IGA2DBrep.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each IGA 2D Brep in model m:

```
var a = IGA2DBrep.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a IGA 2D Brep.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the IGA 2D Brep ig2b:

```
var comm_array = ig2b.GetComments();
```

GetData(index*[integer]*)

Description

Returns the data for fid in *IGA_2D_BREP.

Arguments

- **index** (integer)

Index you want the data for. **Note that indices start at 0**.

Returns

The ID of parametric face, *IGA_FACE_UVW.

Return type

Number

Example

To get the 3rd data entry in IGA 2d brep ig2b:

```
var data = ig2b.GetData(2);
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of IGA2DBrep objects for all of the flagged IGA 2D Breps in a model in PRIMER If the optional property argument is not given then an array of IGA2DBrep objects is returned. If the property argument is given, then that property value for each IGA 2D Brep is returned in the array instead of a IGA2DBrep object

Arguments

- **Model** (Model)

Model to get IGA 2D Breps from

- **flag** (Flag)

Flag set on the IGA 2D Breps that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged IGA 2D Breps in the model

Returns

Array of IGA2DBrep objects or properties

Return type

Array

Example

To make an array of IGA2DBrep objects for all of the IGA 2D Breps in model m flagged with f:

```
var ig2b = IGA2DBrep.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the IGA 2D Breps in model m flagged with f:

```
var a = IGA2DBrep.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the IGA2DBrep object for a IGA 2D Brep ID.

Arguments

- **Model** (Model)

Model to find the IGA 2D Brep in

- **number** (integer)

number of the IGA 2D Brep you want the IGA2DBrep object for

Returns

IGA2DBrep object (or null if IGA 2D Brep does not exist).

Return type

IGA2DBrep

Example

To get the IGA2DBrep object for IGA 2D Brep 100 in model m

```
var ig2b = IGA2DBrep.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a IGA2DBrep property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the IGA2DBrep.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

IGA 2D Brep property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if IGA2DBrep property ig2b.example is a parameter:

```
Options.property_parameter_names = true;
if (ig2b.GetParameter(ig2b.example) ) do_something...
Options.property_parameter_names = false;
```

To check if IGA2DBrep property ig2b.example is a parameter by using the GetParameter method:

```
if (ig2b.ViewParameters().GetParameter(ig2b.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this IGA 2d brep (*IGA_2D_BREP). **Note that a carriage return is not added**. See also IGA2DBrep.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for IGA 2d brep ig2b:

```
var key = ig2b.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the IGA 2d brep. **Note that a carriage return is not added**. See also IGA2DBrep.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for IGA 2d brep ig2b:

```
var cards = ig2b.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last IGA 2D Brep in the model.

Arguments

- **Model** (Model)

Model to get last IGA 2D Brep in

Returns

IGA2DBrep object (or null if there are no IGA 2D Breps in the model).

Return type

IGA2DBrep

Example

To get the last IGA 2D Brep in model m:

```
var ig2b = IGA2DBrep.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free IGA 2D Brep label in the model. Also see IGA2DBrep.FirstFreeLabel(), IGA2DBrep.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free IGA 2D Brep label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

IGA2DBrep label.

Return type

Number

Example

To get the last free IGA 2D Brep label in model m:

```
var label = IGA2DBrep.LastFreeLabel(m);
```

Next()

Description

Returns the next IGA 2D Brep in the model.

Arguments

No arguments

Returns

IGA2DBrep object (or null if there are no more IGA 2D Breps in the model).

Return type

IGA2DBrep

Example

To get the IGA 2D Brep in model m after IGA 2D Brep ig2b:

```
var ig2b = ig2b.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) IGA 2D Brep label in the model. Also see IGA2DBrep.FirstFreeLabel(), IGA2DBrep.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free IGA 2D Brep label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

IGA2DBrep label.

Return type

Number

Example

To get the next free IGA 2D Brep label in model m:

```
var label = IGA2DBrep.NextFreeLabel(m);
```

Previous()

Description

Returns the previous IGA 2D Brep in the model.

Arguments

No arguments

Returns

IGA2DBrep object (or null if there are no more IGA 2D Breps in the model).

Return type

IGA2DBrep

Example

To get the IGA 2D Brep in model m before IGA 2D Brep ig2b:

```
var ig2b = ig2b.Previous();
```

RemoveData(index*[integer]*)

Description

Removes fid for an index in *IGA_2D_BREP.

Arguments

- **index** (integer)

The index you want to delete fid for. **Note that indices start at 0, not 1**.

Returns

No return value.

Example

To delete the fid for the 3rd index for *IGA_2D_BREP in model m:

```
if (ig2b.entries >= 3)
{
    ig2b.RemoveData(2);
}
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the IGA 2D Breps in the model.

Arguments

- **Model** (Model)

Model that all IGA 2D Breps will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the IGA 2D Breps in model m, from 1000000:

```
IGA2DBrep.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged IGA 2D Breps in the model.

Arguments

- **Model** (Model)

Model that all the flagged IGA 2D Breps will be renumbered in

- **flag** (Flag)

Flag set on the IGA 2D Breps that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the IGA 2D Breps in model m flagged with f, from 1000000:

```
IGA2DBrep.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select IGA 2D Breps using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting IGA 2D Breps

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only IGA 2D Breps from that model can be selected. If the argument is a Flag then only IGA 2D Breps that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any IGA 2D Breps can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of IGA 2D Breps selected or null if menu cancelled

Return type

Number

Example

To select IGA 2D Breps from model m, flagging those selected with flag f, giving the prompt 'Select IGA 2D Breps':

```
IGA2DBrep.Select(f, 'Select IGA 2D Breps', m);
```

To select IGA 2D Breps, flagging those selected with flag f but limiting selection to IGA 2D Breps flagged with flag l, giving the prompt 'Select IGA 2D Breps':

```
IGA2DBrep.Select(f, 'Select IGA 2D Breps', l);
```

SetData(index*[integer]*, fid*[integer]*)

Description

Sets fid in *IGA_2D_BREP.

Arguments

- **index** (integer)

Index you want to set the fid for. **Note that indices start at 0**.

- **fid** (integer)

The ID of parametric face.

Returns

No return value.

Example

To add an entry to the *IGA_2D_BREP ig2b with id 10 by using "entries" property to get the next available index:

```
var i = ig2b.entries;
ig2b.SetData(i, 10);
```

To change an entry in 3rd field for *IGA_2D_BREP ig2b with id 30:

```
ig2b.SetData(2, 30);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the IGA 2D Brep.

Arguments

- **flag** (Flag)

Flag to set on the IGA 2D Brep

Returns

No return value

Example

To set flag f for IGA 2D Brep ig2b:

```
ig2b.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the IGA 2D Brep. The IGA 2D Brep will be sketched until you either call IGA2DBrep.Unsketch(), IGA2DBrep.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 2D Brep is sketched. If omitted redraw is true. If you want to sketch several IGA 2D Breps and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch IGA 2D Brep ig2b:

```
ig2b.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged IGA 2D Breps in the model. The IGA 2D Breps will be sketched until you either call IGA2DBrep.Unsketch(), IGA2DBrep.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged IGA 2D Breps will be sketched in

- **flag** (Flag)

Flag set on the IGA 2D Breps that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 2D Breps are sketched. If omitted redraw is true. If you want to sketch flagged IGA 2D Breps several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all IGA 2D Breps flagged with flag in model m:

```
IGA2DBrep.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of IGA 2D Breps in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing IGA 2D Breps should be counted. If false or omitted referenced but undefined IGA 2D Breps will also be included in the total.

Returns

number of IGA 2D Breps

Return type

Number

Example

To get the total number of IGA 2D Breps in model m:

```
var total = IGA2DBrep.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the IGA 2D Breps in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all IGA 2D Breps will be unset in

- **flag** (Flag)

Flag to unset on the IGA 2D Breps

Returns

No return value

Example

To unset the flag f on all the IGA 2D Breps in model m:

```
IGA2DBrep.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the IGA 2D Brep.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 2D Brep is unsketched. If omitted redraw is true. If you want to unsketch several IGA 2D Breps and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch IGA 2D Brep ig2b:

```
ig2b.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all IGA 2D Breps.

Arguments

- **Model** (Model)

Model that all IGA 2D Breps will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 2D Breps are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all IGA 2D Breps in model m:

```
IGA2DBrep.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged IGA 2D Breps in the model.

Arguments

- **Model** (Model)

Model that all IGA 2D Breps will be unsketched in

- **flag** (Flag)

Flag set on the IGA 2D Breps that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 2D Breps are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all IGA 2D Breps flagged with flag in model m:

```
IGA2DBrep.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

IGA2DBrep object.

Return type

IGA2DBrep

Example

To check if IGA2DBrep property ig2b.example is a parameter by using the IGA2DBrep.GetParameter() method:

```
if (ig2b.ViewParameters().GetParameter(ig2b.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for IGA 2D Brep with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA 2D Brep object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...

autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for IGA 2D Brep ig2b:

```
ig2b.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for IGA 2D Brep ig2b:

```
ig2b.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for IGA 2D Brep. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for IGA 2D Brep ig2b:

```
ig2b.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this IGA 2D Brep.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for IGA 2D Brep ig2b:

```
var xrefs = ig2b.Xrefs();
```

---

toString()

Description

Creates a string containing the IGA 2d brep data in keyword format. Note that this contains the keyword header and the keyword cards. See also IGA2DBrep.Keyword() and IGA2DBrep.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for IGA 2d brep ig2b in keyword format

```
var s = ig2b.toString();
```

IGA2DNurbsUVW class

The IGA2DNurbsUVW class gives you access to IGA 2d nurbs uvw cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetControlPoint(index*[integer]*)
- GetParameter(prop*[string]*)
- GetRKnotVector(index*[integer]*)
- GetSKnotVector(index*[integer]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetControlPoint(index*[integer]*, data*[Array of data]*)
- SetFlag(flag*[Flag]*)
- SetRKnotVector(index*[integer]*, r*[real]*)
- SetSKnotVector(index*[integer]*, r*[real]*)
- Sketch(redraw (optional)*[boolean]*)
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

IGA2DNurbsUVW properties

Name	Type	Description
exists (read only)	logical	true if IGA 2D Nurbs UVW exists, false if referred to but not defined.
include	integer	The Include file number that the IGA 2D Nurbs UVW is in.
label	integer	IGA2DNurbsUVW number. Also see the patchid property which is an alternative name for this.
model (read only)	integer	The Model number that the IGA 2D Nurbs UVW is in.
nr	integer	Number of control points in the local r-direction.
ns	integer	Number of control points in the local s-direction.
patchid	integer	IGA2DNurbsUVW number. Also see the label property which is an alternative name for this.
pr	integer	Polynomial degree of the basis in the local r-direction.
ps	integer	Polynomial degree of the basis in the local s-direction.
rfirst	real	First knot value in the local r-direction.
rlast	real	Last knot value in the local r-direction.
sfirst	real	First knot value in the local s-direction.
slast	real	Last knot value in the local s-direction.
unir	integer	Knot vector type in the local r-direction.
unis	integer	Knot vector type in the local s-direction.

Detailed Description

The IGA2DNurbsUVW class allows you to create, modify, edit and manipulate IGA 2d nurbs uvw cards. See the documentation below for more details.

Constructor

new IGA2DNurbsUVW(Model*[Model]*, details*[object]*)

Description

Create a new IGA2DNurbsUVW object.

Arguments

- **Model** (Model)

Model that IGA 2d nurbs uvw will be created in

- **details** (object)

Details for creating the IGA2DNurbsUVW

Object has the following properties:

Name	Type	Description
nr (optional)	integer	Number of control points in the local r-direction.
ns (optional)	integer	Number of control points in the local s-direction.
patchid	integer	ID of the IGA 2D Nurbs UVW.
pr (optional)	integer	Polynomial degree of the basis in the local r-direction.

ps (optional)	integer	Polynomial degree of the basis in the local s-direction.
unir (optional)	integer	Knot vector type in the local r-direction.
unis (optional)	integer	Knot vector type in the local s-direction.

Returns

IGA2DNurbsUVW object

Return type

IGA2DNurbsUVW

Example

To create a new IGA 2d nurbs uvw in model m of id 10:

```
var ig2u = new IGA2DNurbsUVW(m, {patchid:10});
```

To create a new IGA 2d nurbs uvw in model m of id 11 with nr, ns, pr, ps, unir, unis as 12, 13, 14, 15, 21, 22 respectively:

```
var ig2u = new IGA2DNurbsUVW(m, {patchid:11, nr:12, ns:13, pr:14, ps:15,
unir:21, unis:22});
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a IGA 2D Nurbs UVW.

Arguments

- **Comment** (Comment)

Comment that will be attached to the IGA 2D Nurbs UVW

Returns

No return value

Example

To associate comment c to the IGA 2D Nurbs UVW ig2u:

```
ig2u.AssociateComment(c);
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse IGA 2D Nurbs UVW ig2u:

```
ig2u.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the IGA 2D Nurbs UVW.

Arguments

- **flag** (Flag)

Flag to clear on the IGA 2D Nurbs UVW

Returns

No return value

Example

To clear flag f for IGA 2D Nurbs UVW ig2u:

```
ig2u.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the IGA 2D Nurbs UVW. The target include of the copied IGA 2D Nurbs UVW can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

IGA2DNurbsUVW object

Return type

IGA2DNurbsUVW

Example

To copy IGA 2D Nurbs UVW ig2u into IGA 2D Nurbs UVW z:

```
var z = ig2u.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a IGA 2D Nurbs UVW

Arguments

- **Model** (Model)

Model that the IGA 2D Nurbs UVW will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

IGA2DNurbsUVW object (or null if not made).

Return type

IGA2DNurbsUVW

Example

To start creating a IGA 2D Nurbs UVW in model m:

```
var ig2u = IGA2DNurbsUVW.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a IGA 2D Nurbs UVW.

Arguments

- **Comment** (Comment)

Comment that will be detached from the IGA 2D Nurbs UVW

Returns

No return value

Example

To detach comment c from the IGA 2D Nurbs UVW ig2u:

```
ig2u.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit IGA 2D Nurbs UVW ig2u:

```
ig2u.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for IGA 2D Nurbs UVW with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA 2D Nurbs UVW object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for IGA 2D Nurbs UVW ig2u:

```
ig2u.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for IGA 2D Nurbs UVW ig2u:

```
ig2u.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for IGA 2D Nurbs UVW. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for IGA 2D Nurbs UVW ig2u:

```
ig2u.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first IGA 2D Nurbs UVW in the model.

Arguments

- **Model** (Model)

Model to get first IGA 2D Nurbs UVW in

Returns

IGA2DNurbsUVW object (or null if there are no IGA 2D Nurbs UVWs in the model).

Return type

IGA2DNurbsUVW

Example

To get the first IGA 2D Nurbs UVW in model m:

```
var ig2u = IGA2DNurbsUVW.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free IGA 2D Nurbs UVW label in the model. Also see IGA2DNurbsUVW.LastFreeLabel(), IGA2DNurbsUVW.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free IGA 2D Nurbs UVW label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

IGA2DNurbsUVW label.

Return type

Number

Example

To get the first free IGA 2D Nurbs UVW label in model m:

```
var label = IGA2DNurbsUVW.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the IGA 2D Nurbs UVWs in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all IGA 2D Nurbs UVWs will be flagged in

- **flag** (Flag)

Flag to set on the IGA 2D Nurbs UVWs

Returns

No return value

Example

To flag all of the IGA 2D Nurbs UVWs with flag f in model m:

```
IGA2DNurbsUVW.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the IGA 2D Nurbs UVW is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the IGA 2D Nurbs UVW

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if IGA 2D Nurbs UVW ig2u has flag f set on it:

```
if (ig2u.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each IGA 2D Nurbs UVW in the model.
Note that ForEach has been designed to make looping over IGA 2D Nurbs UVWs as fast as possible and so has some limitations.
Firstly, a single temporary IGA2DNurbsUVW object is created and on each function call it is updated with the current IGA 2D Nurbs UVW data. This means that you should not try to store the IGA2DNurbsUVW object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new IGA 2D Nurbs UVWs inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all IGA 2D Nurbs UVWs are in

- **func** (function)

Function to call for each IGA 2D Nurbs UVW

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the IGA 2D Nurbs UVWs in model m:

```
IGA2DNurbsUVW.ForEach(m, test);
function test(ig2u)
{
// ig2u is IGA2DNurbsUVW object
}
```

To call function test for all of the IGA 2D Nurbs UVWs in model m with optional object:

```
var data = { x:0, y:0 };
IGA2DNurbsUVW.ForEach(m, test, data);
function test(ig2u, extra)
{
// ig2u is IGA2DNurbsUVW object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of IGA2DNurbsUVW objects or properties for all of the IGA 2D Nurbs UVWs in a model in PRIMER. If the optional property argument is not given then an array of IGA2DNurbsUVW objects is returned. If the property argument is given, that property value for each IGA 2D Nurbs UVW is returned in the array instead of a IGA2DNurbsUVW object

Arguments

- **Model** (Model)

Model to get IGA 2D Nurbs UVWs from

- **property (optional)** (string)

Name for property to get for all IGA 2D Nurbs UVWs in the model

Returns

Array of IGA2DNurbsUVW objects or properties

Return type

Array

Example

To make an array of IGA2DNurbsUVW objects for all of the IGA 2D Nurbs UVWs in model m:

```
var a = IGA2DNurbsUVW.GetAll(m);
```

To return an array containing the value of property ’foo’ (for example ’x’ for a node) for each IGA 2D Nurbs UVW in model m:

```
var a = IGA2DNurbsUVW.GetAll(m, ’foo’);
```

GetComments()

Description

Extracts the comments associated to a IGA 2D Nurbs UVW.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the IGA 2D Nurbs UVW ig2u:

```
var comm_array = ig2u.GetComments();
```

GetControlPoint(index*[integer]*)

Description

Returns the data for a specific control point an an array. There are nr x ns control points.

Arguments

- **index** (integer)

Index you want the control data for. **Note that indices start at 0**.

Returns

An array containing the control point data.

Return type

Array

Example

To get the data for the 3rd control point in IGA 2d nurbs uvw ig2u:

```
var data = ig2u.GetControlPoint(2);
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of IGA2DNurbsUVW objects for all of the flagged IGA 2D Nurbs UVWs in a model in PRIMER If the optional property argument is not given then an array of IGA2DNurbsUVW objects is returned. If the property argument is given, then that property value for each IGA 2D Nurbs UVW is returned in the array instead of a IGA2DNurbsUVW object

Arguments

- **Model** (Model)

Model to get IGA 2D Nurbs UVWs from

- **flag** (Flag)

Flag set on the IGA 2D Nurbs UVWs that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged IGA 2D Nurbs UVWs in the model

Returns

Array of IGA2DNurbsUVW objects or properties

Return type

Array

Example

To make an array of IGA2DNurbsUVW objects for all of the IGA 2D Nurbs UVWs in model m flagged with f:

```
var ig2u = IGA2DNurbsUVW.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the IGA 2D Nurbs UVWs in model m flagged with f:

```
var a = IGA2DNurbsUVW.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the IGA2DNurbsUVW object for a IGA 2D Nurbs UVW ID.

Arguments

- **Model** (Model)

Model to find the IGA 2D Nurbs UVW in

- **number** (integer)

number of the IGA 2D Nurbs UVW you want the IGA2DNurbsUVW object for

Returns

IGA2DNurbsUVW object (or null if IGA 2D Nurbs UVW does not exist).

Return type

IGA2DNurbsUVW

Example

To get the IGA2DNurbsUVW object for IGA 2D Nurbs UVW 100 in model m

```
var ig2u = IGA2DNurbsUVW.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a IGA2DNurbsUVW property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the IGA2DNurbsUVW.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

IGA 2D Nurbs UVW property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if IGA2DNurbsUVW property ig2u.example is a parameter:

```
Options.property_parameter_names = true;
if (ig2u.GetParameter(ig2u.example) ) do_something...
Options.property_parameter_names = false;
```

To check if IGA2DNurbsUVW property ig2u.example is a parameter by using the GetParameter method:

```
if (ig2u.ViewParameters().GetParameter(ig2u.example) ) do_something...
```

GetRKnotVector(index*[integer]*)

Description

The data for Knot values in the local r-direction.

Arguments

- **index** (integer)

Index you want the r-knot value for. **Note that indices start at 0**.

Returns

Knot values in the local r-direction.

Return type

Number

Example

To get the 3rd r-knot value in IGA 2d nurbs uvw ig2u:

```
var data = ig2u.GetRKnotVector(2);
```

GetSKnotVector(index*[integer]*)

Description

The data for Knot values in the local r-direction.

Arguments

- **index** (integer)

Index you want the s-knot value for. **Note that indices start at 0**.

Returns

Knot values in the local s-direction.

Return type

Number

Example

To get the 3rd s-knot value in IGA 2d nurbs uvw ig2u:

```
var data = ig2u.GetSKnotVector(2);
```

Keyword()

Description

Returns the keyword for this IGA 2d nurbs uvw (*IGA_2D_NURBS_UVW). **Note that a carriage return is not added**. See also IGA2DNurbsUVW.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for IGA 2d nurbs uvw ig2u:

```
var key = ig2u.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the IGA 2d nurbs uvw. **Note that a carriage return is not added**. See also IGA2DNurbsUVW.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for IGA 2d nurbs uvw ig2u:

```
var cards = ig2u.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last IGA 2D Nurbs UVW in the model.

Arguments

- **Model** (Model)

Model to get last IGA 2D Nurbs UVW in

Returns

IGA2DNurbsUVW object (or null if there are no IGA 2D Nurbs UVWs in the model).

Return type

IGA2DNurbsUVW

Example

To get the last IGA 2D Nurbs UVW in model m:

```
var ig2u = IGA2DNurbsUVW.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free IGA 2D Nurbs UVW label in the model. Also see IGA2DNurbsUVW.FirstFreeLabel(), IGA2DNurbsUVW.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free IGA 2D Nurbs UVW label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

IGA2DNurbsUVW label.

Return type

Number

Example

To get the last free IGA 2D Nurbs UVW label in model m:

```
var label = IGA2DNurbsUVW.LastFreeLabel(m);
```

Next()

Description

Returns the next IGA 2D Nurbs UVW in the model.

Arguments

No arguments

Returns

IGA2DNurbsUVW object (or null if there are no more IGA 2D Nurbs UVWs in the model).

Return type

IGA2DNurbsUVW

Example

To get the IGA 2D Nurbs UVW in model m after IGA 2D Nurbs UVW ig2u:

```
var ig2u = ig2u.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) IGA 2D Nurbs UVW label in the model. Also see IGA2DNurbsUVW.FirstFreeLabel(), IGA2DNurbsUVW.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free IGA 2D Nurbs UVW label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

IGA2DNurbsUVW label.

Return type

Number

Example

To get the next free IGA 2D Nurbs UVW label in model m:

```
var label = IGA2DNurbsUVW.NextFreeLabel(m);
```

Previous()

Description

Returns the previous IGA 2D Nurbs UVW in the model.

Arguments

No arguments

Returns

IGA2DNurbsUVW object (or null if there are no more IGA 2D Nurbs UVWs in the model).

Return type

IGA2DNurbsUVW

Example

To get the IGA 2D Nurbs UVW in model m before IGA 2D Nurbs UVW ig2u:

```
var ig2u = ig2u.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the IGA 2D Nurbs UVWs in the model.

Arguments

- **Model** (Model)

Model that all IGA 2D Nurbs UVWs will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the IGA 2D Nurbs UVWs in model m, from 1000000:

```
IGA2DNurbsUVW.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged IGA 2D Nurbs UVWs in the model.

Arguments

- **Model** (Model)

Model that all the flagged IGA 2D Nurbs UVWs will be renumbered in

- **flag** (Flag)

Flag set on the IGA 2D Nurbs UVWs that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the IGA 2D Nurbs UVWs in model m flagged with f, from 1000000:

```
IGA2DNurbsUVW.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select IGA 2D Nurbs UVWs using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting IGA 2D Nurbs UVWs

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only IGA 2D Nurbs UVWs from that model can be selected. If the argument is a Flag then only IGA 2D Nurbs UVWs that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any IGA 2D Nurbs UVWs can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of IGA 2D Nurbs UVWs selected or null if menu cancelled

Return type

Number

Example

To select IGA 2D Nurbs UVWs from model m, flagging those selected with flag f, giving the prompt 'Select IGA 2D Nurbs UVWs':

```
IGA2DNurbsUVW.Select(f, 'Select IGA 2D Nurbs UVWs', m);
```

To select IGA 2D Nurbs UVWs, flagging those selected with flag f but limiting selection to IGA 2D Nurbs UVWs flagged with flag l, giving the prompt 'Select IGA 2D Nurbs UVWs':

```
IGA2DNurbsUVW.Select(f, 'Select IGA 2D Nurbs UVWs', l);
```

SetControlPoint(index*[integer]*, data*[Array of data]*)

Description

Set the data for a specific control point. For each control point there will be control poin U, V, W and control weight WGT. There are nr x ns control points.

Arguments

- **index** (integer)

Index you want the control point data for. **Note that indices start at 0**.

- **data** (Array of data)

Array containing the control point data. The array length should be 4.

Returns

No return value.

Example

To set the 3rd control point data for IGA 2d nurbs uvw ig2u to the values in array data:

```
ig2u.SetControlPoint(2, data);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the IGA 2D Nurbs UVW.

Arguments

- **flag** (Flag)

Flag to set on the IGA 2D Nurbs UVW

Returns

No return value

Example

To set flag f for IGA 2D Nurbs UVW ig2u:

```
ig2u.SetFlag(f);
```

SetRKnotVector(index*[integer]*, r*[real]*)

Description

Sets knot value, r in *IGA_2D_NURBS_UVW. There are [nr + pr + 1] knot values.

Arguments

- **index** (integer)

Index you want to set the r knot vector for. **Note that indices start at 0**.

- **r** (real)

Knot values in the local r-direction

Returns

No return value.

Example

To set the 3rd knot value for IGA 2d nurbs uvw ig2u as 13.5:

```
ig2u.SetRKnotVector(2, 13.5);
```

SetSKnotVector(index*[integer]*, r*[real]*)

Description

Sets knot value, s in *IGA_2D_NURBS_UVW. There are [ns + ps + 1] knot values.

Arguments

- **index** (integer)

Index you want to set the s knot vector for. **Note that indices start at 0**.

- **r** (real)

Knot values in the local s-direction

Returns

No return value.

Example

To set the 2rd knot value for IGA 2d nurbs uvw ig2u as 24.5:

```
ig2u.SetSKnotVector(1, 24.5);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the IGA 2D Nurbs UVW. The IGA 2D Nurbs UVW will be sketched until you either call IGA2DNurbsUVW.Unsketch(), IGA2DNurbsUVW.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 2D Nurbs UVW is sketched. If omitted redraw is true. If you want to sketch several IGA 2D Nurbs UVWs and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch IGA 2D Nurbs UVW ig2u:

```
ig2u.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged IGA 2D Nurbs UVWs in the model. The IGA 2D Nurbs UVWs will be sketched until you either call IGA2DNurbsUVW.Unsketch(), IGA2DNurbsUVW.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged IGA 2D Nurbs UVWs will be sketched in

- **flag** (Flag)

Flag set on the IGA 2D Nurbs UVWs that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 2D Nurbs UVWs are sketched. If omitted redraw is true. If you want to sketch flagged IGA 2D Nurbs UVWs several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all IGA 2D Nurbs UVWs flagged with flag in model m:

```
IGA2DNurbsUVW.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of IGA 2D Nurbs UVWs in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing IGA 2D Nurbs UVWs should be counted. If false or omitted referenced but undefined IGA 2D Nurbs UVWs will also be included in the total.

Returns

number of IGA 2D Nurbs UVWs

Return type

Number

Example

To get the total number of IGA 2D Nurbs UVWs in model m:

```
var total = IGA2DNurbsUVW.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the IGA 2D Nurbs UVWs in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all IGA 2D Nurbs UVWs will be unset in

- **flag** (Flag)

Flag to unset on the IGA 2D Nurbs UVWs

Returns

No return value

Example

To unset the flag f on all the IGA 2D Nurbs UVWs in model m:

```
IGA2DNurbsUVW.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the IGA 2D Nurbs UVW.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 2D Nurbs UVW is unsketched. If omitted redraw is true. If you want to unsketch several IGA 2D Nurbs UVWs and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch IGA 2D Nurbs UVW ig2u:

```
ig2u.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all IGA 2D Nurbs UVWs.

Arguments

- **Model** (Model)

Model that all IGA 2D Nurbs UVWs will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 2D Nurbs UVWs are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all IGA 2D Nurbs UVWs in model m:

```
IGA2DNurbsUVW.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged IGA 2D Nurbs UVWs in the model.

Arguments

- **Model** (Model)

Model that all IGA 2D Nurbs UVWs will be unsketched in

- **flag** (Flag)

Flag set on the IGA 2D Nurbs UVWs that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 2D Nurbs UVWs are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all IGA 2D Nurbs UVWs flagged with flag in model m:

```
IGA2DNurbsUVW.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

IGA2DNurbsUVW object.

Return type

IGA2DNurbsUVW

Example

To check if IGA2DNurbsUVW property ig2u.example is a parameter by using the IGA2DNurbsUVW.GetParameter() method:

```
if (ig2u.ViewParameters().GetParameter(ig2u.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for IGA 2D Nurbs UVW with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA 2D Nurbs UVW object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for IGA 2D Nurbs UVW ig2u:

```
ig2u.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for IGA 2D Nurbs UVW ig2u:

```
ig2u.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for IGA 2D Nurbs UVW. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for IGA 2D Nurbs UVW ig2u:

```
ig2u.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this IGA 2D Nurbs UVW.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for IGA 2D Nurbs UVW ig2u:

```
var xrefs = ig2u.Xrefs();
```

---

toString()

Description

Creates a string containing the IGA 2d nurbs uvw data in keyword format. Note that this contains the keyword header and the keyword cards. See also IGA2DNurbsUVW.Keyword() and IGA2DNurbsUVW.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for IGA 2d nurbs uvw ig2u in keyword format

```
var s = ig2u.toString();
```

IGA2DNurbsXYZ class

The IGA2DNurbsXYZ class gives you access to IGA 2d nurbs xyz cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetControlPoint(index*[integer]*)
- GetParameter(prop*[string]*)
- GetRKnotVector(index*[integer]*)
- GetSKnotVector(index*[integer]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetControlPoint(index*[integer]*, data*[Array of data]*)
- SetFlag(flag*[Flag]*)
- SetRKnotVector(index*[integer]*, r*[real]*)
- SetSKnotVector(index*[integer]*, r*[real]*)
- Sketch(redraw (optional)*[boolean]*)
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

IGA2DNurbsXYZ properties

Name	Type	Description
exists (read only)	logical	true if IGA 2D Nurbs XYZ exists, false if referred to but not defined.
include	integer	The Include file number that the IGA 2D Nurbs XYZ is in.
label	integer	IGA2DNurbsXYZ number. Also see the patchid property which is an alternative name for this.
model (read only)	integer	The Model number that the IGA 2D Nurbs XYZ is in.
nr	integer	Number of control points in the local r-direction.
ns	integer	Number of control points in the local s-direction.
patchid	integer	IGA2DNurbsXYZ number. Also see the label property which is an alternative name for this.
pr	integer	Polynomial degree of the basis in the local r-direction.
ps	integer	Polynomial degree of the basis in the local s-direction.
rfirst	real	First knot value in the local r-direction.
rlast	real	Last knot value in the local r-direction.
sfirst	real	First knot value in the local s-direction.
slast	real	Last knot value in the local s-direction.
unir	integer	Knot vector type in the local r-direction.
unis	integer	Knot vector type in the local s-direction.

Detailed Description

The IGA2DNurbsXYZ class allows you to create, modify, edit and manipulate IGA 2d nurbs xyz cards. See the documentation below for more details.

Constructor

new IGA2DNurbsXYZ(Model*[Model]*, details*[object]*)

Description

Create a new IGA2DNurbsXYZ object.

Arguments

- **Model** (Model)

Model that IGA 2d nurbs xyz will be created in

- **details** (object)

Details for creating the IGA2DNurbsXYZ

Object has the following properties:

Name	Type	Description
nr (optional)	integer	Number of control points in the local r-direction.
ns (optional)	integer	Number of control points in the local s-direction.
patchid	integer	ID of the IGA 2D Nurbs XYZ.
pr (optional)	integer	Polynomial degree of the basis in the local r-direction.

ps (optional)	integer	Polynomial degree of the basis in the local s-direction.
unir (optional)	integer	Knot vector type in the local r-direction.
unis (optional)	integer	Knot vector type in the local s-direction.

Returns

IGA2DNurbsXYZ object

Return type

IGA2DNurbsXYZ

Example

To create a new IGA 2d nurbs xyz in model m of id 10:

```
var ig2x = new IGA2DNurbsXYZ(m, {patchid:10});
```

To create a new IGA 2d nurbs xyz in model m of id 11 with nr, ns, pr, ps, unir, unis as 12, 13, 14, 15, 21, 22 respectively:

```
var ig2x = new IGA2DNurbsXYZ(m, {patchid:11, nr:12, ns:13, pr:14, ps:15,
unir:21, unis:22});
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a IGA 2D Nurbs XYZ.

Arguments

- **Comment** (Comment)

Comment that will be attached to the IGA 2D Nurbs XYZ

Returns

No return value

Example

To associate comment c to the IGA 2D Nurbs XYZ ig2x:

```
ig2x.AssociateComment(c);
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse IGA 2D Nurbs XYZ ig2x:

```
ig2x.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the IGA 2D Nurbs XYZ.

Arguments

- **flag** (Flag)

Flag to clear on the IGA 2D Nurbs XYZ

Returns

No return value

Example

To clear flag f for IGA 2D Nurbs XYZ ig2x:

```
ig2x.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the IGA 2D Nurbs XYZ. The target include of the copied IGA 2D Nurbs XYZ can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

IGA2DNurbsXYZ object

Return type

IGA2DNurbsXYZ

Example

To copy IGA 2D Nurbs XYZ ig2x into IGA 2D Nurbs XYZ z:

```
var z = ig2x.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a IGA 2D Nurbs XYZ

Arguments

- **Model** (Model)

Model that the IGA 2D Nurbs XYZ will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

IGA2DNurbsXYZ object (or null if not made).

Return type

IGA2DNurbsXYZ

Example

To start creating a IGA 2D Nurbs XYZ in model m:

```
var ig2x = IGA2DNurbsXYZ.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a IGA 2D Nurbs XYZ.

Arguments

- **Comment** (Comment)

Comment that will be detached from the IGA 2D Nurbs XYZ

Returns

No return value

Example

To detach comment c from the IGA 2D Nurbs XYZ ig2x:

```
ig2x.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit IGA 2D Nurbs XYZ ig2x:

```
ig2x.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for IGA 2D Nurbs XYZ with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA 2D Nurbs XYZ object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for IGA 2D Nurbs XYZ ig2x:

```
ig2x.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for IGA 2D Nurbs XYZ ig2x:

```
ig2x.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for IGA 2D Nurbs XYZ. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for IGA 2D Nurbs XYZ ig2x:

```
ig2x.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first IGA 2D Nurbs XYZ in the model.

Arguments

- **Model** (Model)

Model to get first IGA 2D Nurbs XYZ in

Returns

IGA2DNurbsXYZ object (or null if there are no IGA 2D Nurbs XYZs in the model).

Return type

IGA2DNurbsXYZ

Example

To get the first IGA 2D Nurbs XYZ in model m:

```
var ig2x = IGA2DNurbsXYZ.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free IGA 2D Nurbs XYZ label in the model. Also see IGA2DNurbsXYZ.LastFreeLabel(), IGA2DNurbsXYZ.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free IGA 2D Nurbs XYZ label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

IGA2DNurbsXYZ label.

Return type

Number

Example

To get the first free IGA 2D Nurbs XYZ label in model m:

```
var label = IGA2DNurbsXYZ.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the IGA 2D Nurbs XYZs in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all IGA 2D Nurbs XYZs will be flagged in

- **flag** (Flag)

Flag to set on the IGA 2D Nurbs XYZs

Returns

No return value

Example

To flag all of the IGA 2D Nurbs XYZs with flag f in model m:

```
IGA2DNurbsXYZ.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the IGA 2D Nurbs XYZ is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the IGA 2D Nurbs XYZ

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if IGA 2D Nurbs XYZ ig2x has flag f set on it:

```
if (ig2x.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each IGA 2D Nurbs XYZ in the model.
Note that ForEach has been designed to make looping over IGA 2D Nurbs XYZs as fast as possible and so has some limitations.
Firstly, a single temporary IGA2DNurbsXYZ object is created and on each function call it is updated with the current IGA 2D Nurbs XYZ data. This means that you should not try to store the IGA2DNurbsXYZ object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new IGA 2D Nurbs XYZs inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all IGA 2D Nurbs XYZs are in

- **func** (function)

Function to call for each IGA 2D Nurbs XYZ

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the IGA 2D Nurbs XYZs in model m:

```
IGA2DNurbsXYZ.ForEach(m, test);
function test(ig2x)
{
// ig2x is IGA2DNurbsXYZ object
}
```

To call function test for all of the IGA 2D Nurbs XYZs in model m with optional object:

```
var data = { x:0, y:0 };
IGA2DNurbsXYZ.ForEach(m, test, data);
function test(ig2x, extra)
{
// ig2x is IGA2DNurbsXYZ object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of IGA2DNurbsXYZ objects or properties for all of the IGA 2D Nurbs XYZs in a model in PRIMER. If the optional property argument is not given then an array of IGA2DNurbsXYZ objects is returned. If the property argument is given, that property value for each IGA 2D Nurbs XYZ is returned in the array instead of a IGA2DNurbsXYZ object

Arguments

- **Model** (Model)

Model to get IGA 2D Nurbs XYZs from

- **property (optional)** (string)

Name for property to get for all IGA 2D Nurbs XYZs in the model

Returns

Array of IGA2DNurbsXYZ objects or properties

Return type

Array

Example

To make an array of IGA2DNurbsXYZ objects for all of the IGA 2D Nurbs XYZs in model m:

```
var a = IGA2DNurbsXYZ.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each IGA 2D Nurbs XYZ in model m:

```
var a = IGA2DNurbsXYZ.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a IGA 2D Nurbs XYZ.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the IGA 2D Nurbs XYZ ig2x:

```
var comm_array = ig2x.GetComments();
```

GetControlPoint(index*[integer]*)

Description

Returns the data for a specific control point an an array. There are nr x ns control points.

Arguments

- **index** (integer)

Index you want the control data for. **Note that indices start at 0**.

Returns

An array containing the control point data.

Return type

Array

Example

To get the data for the 3rd control point in IGA 2d nurbs xyz ig2x:

```
var data = ig2x.GetControlPoint(2);
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of IGA2DNurbsXYZ objects for all of the flagged IGA 2D Nurbs XYZs in a model in PRIMER If the optional property argument is not given then an array of IGA2DNurbsXYZ objects is returned. If the property argument is given, then that property value for each IGA 2D Nurbs XYZ is returned in the array instead of a IGA2DNurbsXYZ object

Arguments

- **Model** (Model)

Model to get IGA 2D Nurbs XYZs from

- **flag** (Flag)

Flag set on the IGA 2D Nurbs XYZs that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged IGA 2D Nurbs XYZs in the model

Returns

Array of IGA2DNurbsXYZ objects or properties

Return type

Array

Example

To make an array of IGA2DNurbsXYZ objects for all of the IGA 2D Nurbs XYZs in model m flagged with f:

```
var ig2x = IGA2DNurbsXYZ.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the IGA 2D Nurbs XYZs in model m flagged with f:

```
var a = IGA2DNurbsXYZ.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the IGA2DNurbsXYZ object for a IGA 2D Nurbs XYZ ID.

Arguments

- **Model** (Model)

Model to find the IGA 2D Nurbs XYZ in

- **number** (integer)

number of the IGA 2D Nurbs XYZ you want the IGA2DNurbsXYZ object for

Returns

IGA2DNurbsXYZ object (or null if IGA 2D Nurbs XYZ does not exist).

Return type

IGA2DNurbsXYZ

Example

To get the IGA2DNurbsXYZ object for IGA 2D Nurbs XYZ 100 in model m

```
var ig2x = IGA2DNurbsXYZ.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a IGA2DNurbsXYZ property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the IGA2DNurbsXYZ.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

IGA 2D Nurbs XYZ property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if IGA2DNurbsXYZ property ig2x.example is a parameter:

```
Options.property_parameter_names = true;
if (ig2x.GetParameter(ig2x.example) ) do_something...
Options.property_parameter_names = false;
```

To check if IGA2DNurbsXYZ property ig2x.example is a parameter by using the GetParameter method:

```
if (ig2x.ViewParameters().GetParameter(ig2x.example) ) do_something...
```

GetRKnotVector(index*[integer]*)

Description

The data for Knot values in the local r-direction.

Arguments

- **index** (integer)

Index you want the r-knot value for. **Note that indices start at 0**.

Returns

Knot values in the local r-direction.

Return type

Number

Example

To get the 3rd r-knot value in IGA 2d nurbs xyz ig2x:

```
var data = ig2x.GetRKnotVector(2);
```

GetSKnotVector(index*[integer]*)

Description

The data for Knot values in the local s-direction.

Arguments

- **index** (integer)

Index you want the s-knot value for. **Note that indices start at 0**.

Returns

Knot values in the local s-direction.

Return type

Number

Example

To get the 3rd s-knot value in IGA 2d nurbs xyz ig2x:

```
var data = ig2x.GetSKnotVector(2);
```

Keyword()

Description

Returns the keyword for this IGA 2d nurbs xyz (*IGA_2D_NURBS_XYZ). **Note that a carriage return is not added**. See also IGA2DNurbsXYZ.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for IGA 2d nurbs xyz ig2x:

```
var key = ig2x.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the IGA 2d nurbs xyz. **Note that a carriage return is not added**. See also IGA2DNurbsXYZ.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for IGA 2d nurbs xyz ig2x:

```
var cards = ig2x.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last IGA 2D Nurbs XYZ in the model.

Arguments

- **Model** (Model)

Model to get last IGA 2D Nurbs XYZ in

Returns

IGA2DNurbsXYZ object (or null if there are no IGA 2D Nurbs XYZs in the model).

Return type

IGA2DNurbsXYZ

Example

To get the last IGA 2D Nurbs XYZ in model m:

```
var ig2x = IGA2DNurbsXYZ.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free IGA 2D Nurbs XYZ label in the model. Also see IGA2DNurbsXYZ.FirstFreeLabel(), IGA2DNurbsXYZ.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free IGA 2D Nurbs XYZ label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

IGA2DNurbsXYZ label.

Return type

Number

Example

To get the last free IGA 2D Nurbs XYZ label in model m:

```
var label = IGA2DNurbsXYZ.LastFreeLabel(m);
```

Next()

Description

Returns the next IGA 2D Nurbs XYZ in the model.

Arguments

No arguments

Returns

IGA2DNurbsXYZ object (or null if there are no more IGA 2D Nurbs XYZs in the model).

Return type

IGA2DNurbsXYZ

Example

To get the IGA 2D Nurbs XYZ in model m after IGA 2D Nurbs XYZ ig2x:

```
var ig2x = ig2x.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) IGA 2D Nurbs XYZ label in the model. Also see IGA2DNurbsXYZ.FirstFreeLabel(), IGA2DNurbsXYZ.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free IGA 2D Nurbs XYZ label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

IGA2DNurbsXYZ label.

Return type

Number

Example

To get the next free IGA 2D Nurbs XYZ label in model m:

```
var label = IGA2DNurbsXYZ.NextFreeLabel(m);
```

Previous()

Description

Returns the previous IGA 2D Nurbs XYZ in the model.

Arguments

No arguments

Returns

IGA2DNurbsXYZ object (or null if there are no more IGA 2D Nurbs XYZs in the model).

Return type

IGA2DNurbsXYZ

Example

To get the IGA 2D Nurbs XYZ in model m before IGA 2D Nurbs XYZ ig2x:

```
var ig2x = ig2x.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the IGA 2D Nurbs XYZs in the model.

Arguments

- **Model** (Model)

Model that all IGA 2D Nurbs XYZs will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the IGA 2D Nurbs XYZs in model m, from 1000000:

```
IGA2DNurbsXYZ.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged IGA 2D Nurbs XYZs in the model.

Arguments

- **Model** (Model)

Model that all the flagged IGA 2D Nurbs XYZs will be renumbered in

- **flag** (Flag)

Flag set on the IGA 2D Nurbs XYZs that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the IGA 2D Nurbs XYZs in model m flagged with f, from 1000000:

```
IGA2DNurbsXYZ.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select IGA 2D Nurbs XYZs using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting IGA 2D Nurbs XYZs

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only IGA 2D Nurbs XYZs from that model can be selected. If the argument is a Flag then only IGA 2D Nurbs XYZs that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any IGA 2D Nurbs XYZs can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of IGA 2D Nurbs XYZs selected or null if menu cancelled

Return type

Number

Example

To select IGA 2D Nurbs XYZs from model m, flagging those selected with flag f, giving the prompt 'Select IGA 2D Nurbs XYZs':

```
IGA2DNurbsXYZ.Select(f, 'Select IGA 2D Nurbs XYZs', m);
```

To select IGA 2D Nurbs XYZs, flagging those selected with flag f but limiting selection to IGA 2D Nurbs XYZs flagged with flag l, giving the prompt 'Select IGA 2D Nurbs XYZs':

```
IGA2DNurbsXYZ.Select(f, 'Select IGA 2D Nurbs XYZs', l);
```

SetControlPoint(index*[integer]*, data*[Array of data]*)

Description

Set the data for a specific control point. For each control point there will be control point X, Y, Z and control weight WGT. There are nr x ns control points.

Arguments

- **index** (integer)

Index you want the control point data for. **Note that indices start at 0**.

- **data** (Array of data)

Array containing the control point data. The array length should be 4.

Returns

No return value.

Example

To set the 3rd control point data for IGA 2d nurbs xyz ig2x to the values in array data:

```
ig2x.SetControlPoint(2, data);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the IGA 2D Nurbs XYZ.

Arguments

- **flag** (Flag)

Flag to set on the IGA 2D Nurbs XYZ

Returns

No return value

Example

To set flag f for IGA 2D Nurbs XYZ ig2x:

```
ig2x.SetFlag(f);
```

SetRKnotVector(index*[integer]*, r*[real]*)

Description

Sets knot value, r in *IGA_2D_NURBS_XYZ. There are [nr + pr + 1] knot values.

Arguments

- **index** (integer)

Index you want to set the r knot vector for. **Note that indices start at 0**.

- **r** (real)

Knot values in the local r-direction

Returns

No return value.

Example

To set the 3rd knot value for IGA 2d nurbs xyz ig2x as 13.5:

```
ig2x.SetRKnotVector(2, 13.5);
```

SetSKnotVector(index*[integer]*, r*[real]*)

Description

Sets knot value, s in *IGA_2D_NURBS_XYZ. There are [ns + ps + 1] knot values.

Arguments

- **index** (integer)

Index you want to set the s knot vector for. **Note that indices start at 0**.

- **r** (real)

Knot values in the local s-direction

Returns

No return value.

Example

To set the 2rd knot value for IGA 2d nurbs xyz ig2x as 24.5:

```
ig2x.SetSKnotVector(1, 24.5);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the IGA 2D Nurbs XYZ. The IGA 2D Nurbs XYZ will be sketched until you either call IGA2DNurbsXYZ.Unsketch(), IGA2DNurbsXYZ.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 2D Nurbs XYZ is sketched. If omitted redraw is true. If you want to sketch several IGA 2D Nurbs XYZs and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch IGA 2D Nurbs XYZ ig2x:

```
ig2x.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged IGA 2D Nurbs XYZs in the model. The IGA 2D Nurbs XYZs will be sketched until you either call IGA2DNurbsXYZ.Unsketch(), IGA2DNurbsXYZ.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged IGA 2D Nurbs XYZs will be sketched in

- **flag** (Flag)

Flag set on the IGA 2D Nurbs XYZs that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 2D Nurbs XYZs are sketched. If omitted redraw is true. If you want to sketch flagged IGA 2D Nurbs XYZs several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all IGA 2D Nurbs XYZs flagged with flag in model m:

```
IGA2DNurbsXYZ.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of IGA 2D Nurbs XYZs in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing IGA 2D Nurbs XYZs should be counted. If false or omitted referenced but undefined IGA 2D Nurbs XYZs will also be included in the total.

Returns

number of IGA 2D Nurbs XYZs

Return type

Number

Example

To get the total number of IGA 2D Nurbs XYZs in model m:

```
var total = IGA2DNurbsXYZ.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the IGA 2D Nurbs XYZs in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all IGA 2D Nurbs XYZs will be unset in

- **flag** (Flag)

Flag to unset on the IGA 2D Nurbs XYZs

Returns

No return value

Example

To unset the flag f on all the IGA 2D Nurbs XYZs in model m:

```
IGA2DNurbsXYZ.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the IGA 2D Nurbs XYZ.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 2D Nurbs XYZ is unsketched. If omitted redraw is true. If you want to unsketch several IGA 2D Nurbs XYZs and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch IGA 2D Nurbs XYZ ig2x:

```
ig2x.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all IGA 2D Nurbs XYZs.

Arguments

- **Model** (Model)

Model that all IGA 2D Nurbs XYZs will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 2D Nurbs XYZs are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all IGA 2D Nurbs XYZs in model m:

```
IGA2DNurbsXYZ.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged IGA 2D Nurbs XYZs in the model.

Arguments

- **Model** (Model)

Model that all IGA 2D Nurbs XYZs will be unsketched in

- **flag** (Flag)

Flag set on the IGA 2D Nurbs XYZs that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 2D Nurbs XYZs are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all IGA 2D Nurbs XYZs flagged with flag in model m:

```
IGA2DNurbsXYZ.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

IGA2DNurbsXYZ object.

Return type

IGA2DNurbsXYZ

Example

To check if IGA2DNurbsXYZ property ig2x.example is a parameter by using the IGA2DNurbsXYZ.GetParameter() method:

```
if (ig2x.ViewParameters().GetParameter(ig2x.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for IGA 2D Nurbs XYZ with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA 2D Nurbs XYZ object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for IGA 2D Nurbs XYZ ig2x:

```
ig2x.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for IGA 2D Nurbs XYZ ig2x:

```
ig2x.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for IGA 2D Nurbs XYZ. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for IGA 2D Nurbs XYZ ig2x:

```
ig2x.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this IGA 2D Nurbs XYZ.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for IGA 2D Nurbs XYZ ig2x:

```
var xrefs = ig2x.Xrefs();
```

---

toString()

Description

Creates a string containing the IGA 2d nurbs xyz data in keyword format. Note that this contains the keyword header and the keyword cards. See also IGA2DNurbsXYZ.Keyword() and IGA2DNurbsXYZ.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for IGA 2d nurbs xyz ig2x in keyword format

```
var s = ig2x.toString();
```

IGA3DBasisTransformXYZ class

The IGA3DBasisTransformXYZ class gives you access to IGA 3d basis transform xyz cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

IGA3DBasisTransformXYZ properties

Name	Type	Description

exists (read only)	logical	true if IGA 3D Basis Transform XYZ exists, false if referred to but not defined.
filename	string	Number of control points in the local r-direction.
include	integer	The Include file number that the IGA 3D Basis Transform XYZ is in.
label	integer	IGA3DBasisTransformXYZ number. Also see the patchid property which is an alternative name for this.
model (read only)	integer	The Model number that the IGA 3D Basis Transform XYZ is in.
patchid	integer	IGA3DBasisTransformXYZ number. Also see the label property which is an alternative name for this.

Detailed Description

The IGA3DBasisTransformXYZ class allows you to create, modify, edit and manipulate IGA 3d basis transform xyz cards. See the documentation below for more details.

Constructor

new IGA3DBasisTransformXYZ(Model*[Model]*, details*[object]*)

Description

Create a new IGA3DBasisTransformXYZ object.

Arguments

- **Model** (Model)

Model that IGA 3d basis transform xyz will be created in

- **details** (object)

Details for creating the IGA3DBasisTransformXYZ

Object has the following properties:

Name	Type	Description
filename (optional)	string	Name of file containing patch data.
patchid	integer	ID of the IGA 3D Basis Transform XYZ.

Returns

IGA3DBasisTransformXYZ object

Return type

IGA3DBasisTransformXYZ

Example

To create a new IGA 3d basis transform xyz in model m of id 10:

```
var ig3bt = new IGA3DBasisTransformXYZ(m, {patchid:10});
```

To create a new IGA 3d basis transform xyz in model m of id 10 with a filename:

```
var ig3bt = new IGA3DBasisTransformXYZ(m, {patchid:10,
filename:"Filename\\with\\path.extension"});
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a IGA 3D Basis Transform XYZ.

Arguments

- **Comment** (Comment)

Comment that will be attached to the IGA 3D Basis Transform XYZ

Returns

No return value

Example

To associate comment c to the IGA 3D Basis Transform XYZ ig3bt:

```
ig3bt.AssociateComment(c);
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse IGA 3D Basis Transform XYZ ig3bt:

```
ig3bt.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the IGA 3D Basis Transform XYZ.

Arguments

- **flag** (Flag)

Flag to clear on the IGA 3D Basis Transform XYZ

Returns

No return value

Example

To clear flag f for IGA 3D Basis Transform XYZ ig3bt:

```
ig3bt.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the IGA 3D Basis Transform XYZ. The target include of the copied IGA 3D Basis Transform XYZ can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

IGA3DBasisTransformXYZ object

Return type

IGA3DBasisTransformXYZ

Example

To copy IGA 3D Basis Transform XYZ ig3bt into IGA 3D Basis Transform XYZ z:

```
var z = ig3bt.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a IGA 3D Basis Transform XYZ

Arguments

- **Model** (Model)

Model that the IGA 3D Basis Transform XYZ will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

IGA3DBasisTransformXYZ object (or null if not made).

Return type

IGA3DBasisTransformXYZ

Example

To start creating a IGA 3D Basis Transform XYZ in model m:

```
var ig3bt = IGA3DBasisTransformXYZ.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a IGA 3D Basis Transform XYZ.

Arguments

- **Comment** (Comment)

Comment that will be detached from the IGA 3D Basis Transform XYZ

Returns

No return value

Example

To detach comment c from the IGA 3D Basis Transform XYZ ig3bt:

```
ig3bt.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit IGA 3D Basis Transform XYZ ig3bt:

```
ig3bt.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for IGA 3D Basis Transform XYZ with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description

autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA 3D Basis Transform XYZ object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for IGA 3D Basis Transform XYZ ig3bt:

```
ig3bt.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for IGA 3D Basis Transform XYZ ig3bt:

```
ig3bt.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for IGA 3D Basis Transform XYZ. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for IGA 3D Basis Transform XYZ ig3bt:

```
ig3bt.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first IGA 3D Basis Transform XYZ in the model.

Arguments

- **Model** (Model)

Model to get first IGA 3D Basis Transform XYZ in

Returns

IGA3DBasisTransformXYZ object (or null if there are no IGA 3D Basis Transform XYZs in the model).

Return type

IGA3DBasisTransformXYZ

Example

To get the first IGA 3D Basis Transform XYZ in model m:

```
var ig3bt = IGA3DBasisTransformXYZ.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free IGA 3D Basis Transform XYZ label in the model. Also see IGA3DBasisTransformXYZ.LastFreeLabel(), IGA3DBasisTransformXYZ.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free IGA 3D Basis Transform XYZ label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

IGA3DBasisTransformXYZ label.

Return type

Number

Example

To get the first free IGA 3D Basis Transform XYZ label in model m:

```
var label = IGA3DBasisTransformXYZ.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the IGA 3D Basis Transform XYZs in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all IGA 3D Basis Transform XYZs will be flagged in

- **flag** (Flag)

Flag to set on the IGA 3D Basis Transform XYZs

Returns

No return value

Example

To flag all of the IGA 3D Basis Transform XYZs with flag f in model m:

```
IGA3DBasisTransformXYZ.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the IGA 3D Basis Transform XYZ is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the IGA 3D Basis Transform XYZ

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if IGA 3D Basis Transform XYZ ig3bt has flag f set on it:

```
if (ig3bt.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each IGA 3D Basis Transform XYZ in the model.
Note that ForEach has been designed to make looping over IGA 3D Basis Transform XYZs as fast as possible and so has some limitations.
Firstly, a single temporary IGA3DBasisTransformXYZ object is created and on each function call it is updated with the current IGA 3D Basis Transform XYZ data. This means that you should not try to store the IGA3DBasisTransformXYZ object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new IGA 3D Basis Transform XYZs inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all IGA 3D Basis Transform XYZs are in

- **func** (function)

Function to call for each IGA 3D Basis Transform XYZ

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the IGA 3D Basis Transform XYZs in model m:

```
IGA3DBasisTransformXYZ.ForEach(m, test);
function test(ig3bt)
{
// ig3bt is IGA3DBasisTransformXYZ object
}
```

To call function test for all of the IGA 3D Basis Transform XYZs in model m with optional object:

```
var data = { x:0, y:0 };
IGA3DBasisTransformXYZ.ForEach(m, test, data);
function test(ig3bt, extra)
{
// ig3bt is IGA3DBasisTransformXYZ object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of IGA3DBasisTransformXYZ objects or properties for all of the IGA 3D Basis Transform XYZs in a model in PRIMER. If the optional property argument is not given then an array of IGA3DBasisTransformXYZ objects is returned. If the property argument is given, that property value for each IGA 3D Basis Transform XYZ is returned in the array instead of a IGA3DBasisTransformXYZ object

Arguments

- **Model** (Model)

Model to get IGA 3D Basis Transform XYZs from

- **property (optional)** (string)

Name for property to get for all IGA 3D Basis Transform XYZs in the model

Returns

Array of IGA3DBasisTransformXYZ objects or properties

Return type

Array

Example

To make an array of IGA3DBasisTransformXYZ objects for all of the IGA 3D Basis Transform XYZs in model m:

```
var a = IGA3DBasisTransformXYZ.GetAll(m);
```

To return an array containing the value of property ’foo’ (for example ’x’ for a node) for each IGA 3D Basis Transform XYZ in model m:

```
var a = IGA3DBasisTransformXYZ.GetAll(m, ’foo’);
```

GetComments()

Description

Extracts the comments associated to a IGA 3D Basis Transform XYZ.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the IGA 3D Basis Transform XYZ ig3bt:

```
var comm_array = ig3bt.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of IGA3DBasisTransformXYZ objects for all of the flagged IGA 3D Basis Transform XYZs in a model in PRIMER If the optional property argument is not given then an array of IGA3DBasisTransformXYZ objects is returned. If the property argument is given, then that property value for each IGA 3D Basis Transform XYZ is returned in the array instead of a IGA3DBasisTransformXYZ object

Arguments

- **Model** (Model)

Model to get IGA 3D Basis Transform XYZs from

- **flag** (Flag)

Flag set on the IGA 3D Basis Transform XYZs that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged IGA 3D Basis Transform XYZs in the model

Returns

Array of IGA3DBasisTransformXYZ objects or properties

Return type

Array

Example

To make an array of IGA3DBasisTransformXYZ objects for all of the IGA 3D Basis Transform XYZs in model m flagged with f:

```
var ig3bt = IGA3DBasisTransformXYZ.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the IGA 3D Basis Transform XYZs in model m flagged with f:

```
var a = IGA3DBasisTransformXYZ.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the IGA3DBasisTransformXYZ object for a IGA 3D Basis Transform XYZ ID.

Arguments

- **Model** (Model)

Model to find the IGA 3D Basis Transform XYZ in

- **number** (integer)

number of the IGA 3D Basis Transform XYZ you want the IGA3DBasisTransformXYZ object for

Returns

IGA3DBasisTransformXYZ object (or null if IGA 3D Basis Transform XYZ does not exist).

Return type

IGA3DBasisTransformXYZ

Example

To get the IGA3DBasisTransformXYZ object for IGA 3D Basis Transform XYZ 100 in model m

```
var ig3bt = IGA3DBasisTransformXYZ.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a IGA3DBasisTransformXYZ property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the IGA3DBasisTransformXYZ.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

IGA 3D Basis Transform XYZ property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if IGA3DBasisTransformXYZ property ig3bt.example is a parameter:

```
Options.property_parameter_names = true;
if (ig3bt.GetParameter(ig3bt.example) ) do_something...
Options.property_parameter_names = false;
```

To check if IGA3DBasisTransformXYZ property ig3bt.example is a parameter by using the GetParameter method:

```
if (ig3bt.ViewParameters().GetParameter(ig3bt.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this IGA 3d basis transform xyz (*IGA_3D_BASIS_TRANSFORM_XYZ). **Note that a carriage return is not added**. See also IGA3DBasisTransformXYZ.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for IGA 3d basis transform xyz ig3bt:

```
var key = ig3bt.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the IGA 3d basis transform xyz. **Note that a carriage return is not added**. See also IGA3DBasisTransformXYZ.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for IGA 3d basis transform xyz ig3bt:

```
var cards = ig3bt.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last IGA 3D Basis Transform XYZ in the model.

Arguments

- **Model** (Model)

Model to get last IGA 3D Basis Transform XYZ in

Returns

IGA3DBasisTransformXYZ object (or null if there are no IGA 3D Basis Transform XYZs in the model).

Return type

IGA3DBasisTransformXYZ

Example

To get the last IGA 3D Basis Transform XYZ in model m:

```
var ig3bt = IGA3DBasisTransformXYZ.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free IGA 3D Basis Transform XYZ label in the model. Also see IGA3DBasisTransformXYZ.FirstFreeLabel(), IGA3DBasisTransformXYZ.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free IGA 3D Basis Transform XYZ label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

IGA3DBasisTransformXYZ label.

Return type

Number

Example

To get the last free IGA 3D Basis Transform XYZ label in model m:

```
var label = IGA3DBasisTransformXYZ.LastFreeLabel(m);
```

Next()

Description

Returns the next IGA 3D Basis Transform XYZ in the model.

Arguments

No arguments

Returns

IGA3DBasisTransformXYZ object (or null if there are no more IGA 3D Basis Transform XYZs in the model).

Return type

IGA3DBasisTransformXYZ

Example

To get the IGA 3D Basis Transform XYZ in model m after IGA 3D Basis Transform XYZ ig3bt:

```
var ig3bt = ig3bt.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) IGA 3D Basis Transform XYZ label in the model. Also see IGA3DBasisTransformXYZ.FirstFreeLabel(), IGA3DBasisTransformXYZ.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free IGA 3D Basis Transform XYZ label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

IGA3DBasisTransformXYZ label.

Return type

Number

Example

To get the next free IGA 3D Basis Transform XYZ label in model m:

```
var label = IGA3DBasisTransformXYZ.NextFreeLabel(m);
```

Previous()

Description

Returns the previous IGA 3D Basis Transform XYZ in the model.

Arguments

No arguments

Returns

IGA3DBasisTransformXYZ object (or null if there are no more IGA 3D Basis Transform XYZs in the model).

Return type

IGA3DBasisTransformXYZ

Example

To get the IGA 3D Basis Transform XYZ in model m before IGA 3D Basis Transform XYZ ig3bt:

```
var ig3bt = ig3bt.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the IGA 3D Basis Transform XYZs in the model.

Arguments

- **Model** (Model)

Model that all IGA 3D Basis Transform XYZs will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the IGA 3D Basis Transform XYZs in model m, from 1000000:

```
IGA3DBasisTransformXYZ.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged IGA 3D Basis Transform XYZs in the model.

Arguments

- **Model** (Model)

Model that all the flagged IGA 3D Basis Transform XYZs will be renumbered in

- **flag** (Flag)

Flag set on the IGA 3D Basis Transform XYZs that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the IGA 3D Basis Transform XYZs in model m flagged with f, from 1000000:

```
IGA3DBasisTransformXYZ.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select IGA 3D Basis Transform XYZs using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting IGA 3D Basis Transform XYZs

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only IGA 3D Basis Transform XYZs from that model can be selected. If the argument is a Flag then only IGA 3D Basis Transform XYZs that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any IGA 3D Basis Transform XYZs can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of IGA 3D Basis Transform XYZs selected or null if menu cancelled

Return type

Number

Example

To select IGA 3D Basis Transform XYZs from model m, flagging those selected with flag f, giving the prompt 'Select IGA 3D Basis Transform XYZs':

```
IGA3DBasisTransformXYZ.Select(f, 'Select IGA 3D Basis Transform XYZs', m);
```

To select IGA 3D Basis Transform XYZs, flagging those selected with flag f but limiting selection to IGA 3D Basis Transform XYZs flagged with flag l, giving the prompt 'Select IGA 3D Basis Transform XYZs':

```
IGA3DBasisTransformXYZ.Select(f, 'Select IGA 3D Basis Transform XYZs', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the IGA 3D Basis Transform XYZ.

Arguments

- **flag** (Flag)

Flag to set on the IGA 3D Basis Transform XYZ

Returns

No return value

Example

To set flag f for IGA 3D Basis Transform XYZ ig3bt:

```
ig3bt.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the IGA 3D Basis Transform XYZ. The IGA 3D Basis Transform XYZ will be sketched until you either call IGA3DBasisTransformXYZ.Unsketch(), IGA3DBasisTransformXYZ.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 3D Basis Transform XYZ is sketched. If omitted redraw is true. If you want to sketch several IGA 3D Basis Transform XYZs and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch IGA 3D Basis Transform XYZ ig3bt:

```
ig3bt.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged IGA 3D Basis Transform XYZs in the model. The IGA 3D Basis Transform XYZs will be sketched until you either call IGA3DBasisTransformXYZ.Unsketch(), IGA3DBasisTransformXYZ.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged IGA 3D Basis Transform XYZs will be sketched in

- **flag** (Flag)

Flag set on the IGA 3D Basis Transform XYZs that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 3D Basis Transform XYZs are sketched. If omitted redraw is true. If you want to sketch flagged IGA 3D Basis Transform XYZs several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all IGA 3D Basis Transform XYZs flagged with flag in model m:

```
IGA3DBasisTransformXYZ.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of IGA 3D Basis Transform XYZs in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing IGA 3D Basis Transform XYZs should be counted. If false or omitted referenced but undefined IGA 3D Basis Transform XYZs will also be included in the total.

Returns

number of IGA 3D Basis Transform XYZs

Return type

Number

Example

To get the total number of IGA 3D Basis Transform XYZs in model m:

```
var total = IGA3DBasisTransformXYZ.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the IGA 3D Basis Transform XYZs in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all IGA 3D Basis Transform XYZs will be unset in

- **flag** (Flag)

Flag to unset on the IGA 3D Basis Transform XYZs

Returns

No return value

Example

To unset the flag f on all the IGA 3D Basis Transform XYZs in model m:

```
IGA3DBasisTransformXYZ.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the IGA 3D Basis Transform XYZ.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 3D Basis Transform XYZ is unsketched. If omitted redraw is true. If you want to unsketch several IGA 3D Basis Transform XYZs and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch IGA 3D Basis Transform XYZ ig3bt:

```
ig3bt.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all IGA 3D Basis Transform XYZs.

Arguments

- **Model** (Model)

Model that all IGA 3D Basis Transform XYZs will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 3D Basis Transform XYZs are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all IGA 3D Basis Transform XYZs in model m:

```
IGA3DBasisTransformXYZ.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged IGA 3D Basis Transform XYZs in the model.

Arguments

- **Model** (Model)

Model that all IGA 3D Basis Transform XYZs will be unsketched in

- **flag** (Flag)

Flag set on the IGA 3D Basis Transform XYZs that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 3D Basis Transform XYZs are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all IGA 3D Basis Transform XYZs flagged with flag in model m:

```
IGA3DBasisTransformXYZ.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

IGA3DBasisTransformXYZ object.

Return type

IGA3DBasisTransformXYZ

Example

To check if IGA3DBasisTransformXYZ property ig3bt.example is a parameter by using the IGA3DBasisTransformXYZ.GetParameter() method:

```
if (ig3bt.ViewParameters().GetParameter(ig3bt.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for IGA 3D Basis Transform XYZ with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA 3D Basis Transform XYZ object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for IGA 3D Basis Transform XYZ ig3bt:

```
ig3bt.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for IGA 3D Basis Transform XYZ ig3bt:

```
ig3bt.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for IGA 3D Basis Transform XYZ. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for IGA 3D Basis Transform XYZ ig3bt:

```
ig3bt.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this IGA 3D Basis Transform XYZ.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for IGA 3D Basis Transform XYZ ig3bt:

```
var xrefs = ig3bt.Xrefs();
```

---

toString()

Description

Creates a string containing the IGA 3d basis transform xyz data in keyword format. Note that this contains the keyword header and the keyword cards. See also IGA3DBasisTransformXYZ.Keyword() and IGA3DBasisTransformXYZ.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for IGA 3d basis transform xyz ig3bt in keyword format

```
var s = ig3bt.toString();
```

IGA3DNurbsXYZ class

The IGA3DNurbsXYZ class gives you access to IGA 3d nurbs xyz cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetControlPoint(index*[integer]*)
- GetParameter(prop*[string]*)
- GetRKnotVector(index*[integer]*)
- GetSKnotVector(index*[integer]*)
- GetTKnotVector(index*[integer]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetControlPoint(index*[integer]*, data*[Array of data]*)
- SetFlag(flag*[Flag]*)
- SetRKnotVector(index*[integer]*, r*[real]*)
- SetSKnotVector(index*[integer]*, r*[real]*)
- SetTKnotVector(index*[integer]*, r*[real]*)
- Sketch(redraw (optional)*[boolean]*)
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]

- Xrefs()
- toString()

IGA3DNurbsXYZ properties

Name	Type	Description
exists (read only)	logical	true if IGA 3D Nurbs XYZ exists, false if referred to but not defined.
include	integer	The Include file number that the IGA 3D Nurbs XYZ is in.
label	integer	IGA3DNurbsXYZ number. Also see the patchid property which is an alternative name for this.
model (read only)	integer	The Model number that the IGA 3D Nurbs XYZ is in.
nr	integer	Number of control points in the local r-direction.
ns	integer	Number of control points in the local s-direction.
nt	integer	Number of control points in the local t-direction.
patchid	integer	IGA3DNurbsXYZ number. Also see the label property which is an alternative name for this.
pr	integer	Polynomial degree of the basis in the local r-direction.
ps	integer	Polynomial degree of the basis in the local s-direction.
pt	integer	Polynomial degree of the basis in the local t-direction.
rfirst	real	First knot value in the local r-direction.
rlast	real	Last knot value in the local r-direction.
sfirst	real	First knot value in the local s-direction.
slast	real	Last knot value in the local s-direction.
tfirst	real	First knot value in the local t-direction.
tlast	real	Last knot value in the local t-direction.
unir	integer	Knot vector type in the local r-direction.
unis	integer	Knot vector type in the local s-direction.
unit	integer	Knot vector type in the local t-direction.

Detailed Description

The IGA3DNurbsXYZ class allows you to create, modify, edit and manipulate IGA 3d nurbs xyz cards. See the documentation below for more details.

Constructor

new IGA3DNurbsXYZ(Model*[Model]*, details*[object]*)

Description

Create a new IGA3DNurbsXYZ object.

Arguments

- **Model** (Model)

Model that IGA 3d nurbs xyz will be created in

- **details** (object)

Details for creating the IGA3DNurbsXYZ

Object has the following properties:

Name	Type	Description
nr (optional)	integer	Number of control points in the local r-direction.
ns (optional)	integer	Number of control points in the local s-direction.
nt (optional)	integer	Number of control points in the local t-direction.
patchid	integer	ID of the IGA 3D Nurbs XYZ.
pr (optional)	integer	Polynomial degree of the basis in the local r-direction.
ps (optional)	integer	Polynomial degree of the basis in the local s-direction.
pt (optional)	integer	Polynomial degree of the basis in the local t-direction.
unir (optional)	integer	Knot vector type in the local r-direction.
unis (optional)	integer	Knot vector type in the local s-direction.
unit (optional)	integer	Knot vector type in the local t-direction.

Returns

IGA3DNurbsXYZ object

Return type

IGA3DNurbsXYZ

Example

To create a new IGA 3d nurbs xyz in model m of id 10:

```
var ig3x = new IGA3DNurbsXYZ(m, {patchid:10});
```

To create a new IGA 3d nurbs xyz in model m of id 11 with nr, ns, nt, pr, ps, pt, unir, unis, unit as 12, 13, 14, 15, 16, 17, 21, 22, 23 respectively:

```
var ig3x = new IGA3DNurbsXYZ(m, {patchid:11, nr:12, ns:13, nt:14, pr:15, ps:16, pt:17, unir:21, unis:22, unit:23});
```

Details of functions

AssociateComment(Comment[*Comment*])

Description

Associates a comment with a IGA 3D Nurbs XYZ.

Arguments

- **Comment** (Comment)

Comment that will be attached to the IGA 3D Nurbs XYZ

Returns

No return value

Example

To associate comment c to the IGA 3D Nurbs XYZ ig3x:

```
ig3x.AssociateComment(c);
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse IGA 3D Nurbs XYZ ig3x:

```
ig3x.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the IGA 3D Nurbs XYZ.

Arguments

- **flag** (Flag)

Flag to clear on the IGA 3D Nurbs XYZ

Returns

No return value

Example

To clear flag f for IGA 3D Nurbs XYZ ig3x:

```
ig3x.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the IGA 3D Nurbs XYZ. The target include of the copied IGA 3D Nurbs XYZ can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

IGA3DNurbsXYZ object

Return type

IGA3DNurbsXYZ

Example

To copy IGA 3D Nurbs XYZ ig3x into IGA 3D Nurbs XYZ z:

```
var z = ig3x.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a IGA 3D Nurbs XYZ

Arguments

- **Model** (Model)

Model that the IGA 3D Nurbs XYZ will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

IGA3DNurbsXYZ object (or null if not made).

Return type

IGA3DNurbsXYZ

Example

To start creating a IGA 3D Nurbs XYZ in model m:

```
var ig3x = IGA3DNurbsXYZ.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a IGA 3D Nurbs XYZ.

Arguments

- **Comment** (Comment)

Comment that will be detached from the IGA 3D Nurbs XYZ

Returns

No return value

Example

To detach comment c from the IGA 3D Nurbs XYZ ig3x:

```
ig3x.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit IGA 3D Nurbs XYZ ig3x:

```
ig3x.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for IGA 3D Nurbs XYZ with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA 3D Nurbs XYZ object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for IGA 3D Nurbs XYZ ig3x:

```
ig3x.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for IGA 3D Nurbs XYZ ig3x:

```
ig3x.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for IGA 3D Nurbs XYZ. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for IGA 3D Nurbs XYZ ig3x:

```
ig3x.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first IGA 3D Nurbs XYZ in the model.

Arguments

- **Model** (Model)

Model to get first IGA 3D Nurbs XYZ in

Returns

IGA3DNurbsXYZ object (or null if there are no IGA 3D Nurbs XYZs in the model).

Return type

IGA3DNurbsXYZ

Example

To get the first IGA 3D Nurbs XYZ in model m:

```
var ig3x = IGA3DNurbsXYZ.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free IGA 3D Nurbs XYZ label in the model. Also see IGA3DNurbsXYZ.LastFreeLabel(), IGA3DNurbsXYZ.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free IGA 3D Nurbs XYZ label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

IGA3DNurbsXYZ label.

Return type

Number

Example

To get the first free IGA 3D Nurbs XYZ label in model m:

```
var label = IGA3DNurbsXYZ.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the IGA 3D Nurbs XYZs in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all IGA 3D Nurbs XYZs will be flagged in

- **flag** (Flag)

Flag to set on the IGA 3D Nurbs XYZs

Returns

No return value

Example

To flag all of the IGA 3D Nurbs XYZs with flag f in model m:

```
IGA3DNurbsXYZ.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the IGA 3D Nurbs XYZ is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the IGA 3D Nurbs XYZ

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if IGA 3D Nurbs XYZ ig3x has flag f set on it:

```
if (ig3x.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each IGA 3D Nurbs XYZ in the model.
Note that ForEach has been designed to make looping over IGA 3D Nurbs XYZs as fast as possible and so has some limitations.
Firstly, a single temporary IGA3DNurbsXYZ object is created and on each function call it is updated with the current IGA 3D Nurbs XYZ data. This means that you should not try to store the IGA3DNurbsXYZ object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new IGA 3D Nurbs XYZs inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all IGA 3D Nurbs XYZs are in

- **func** (function)

Function to call for each IGA 3D Nurbs XYZ

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the IGA 3D Nurbs XYZs in model m:

```
IGA3DNurbsXYZ.ForEach(m, test);
function test(ig3x)
{
// ig3x is IGA3DNurbsXYZ object
}
```

To call function test for all of the IGA 3D Nurbs XYZs in model m with optional object:

```
var data = { x:0, y:0 };
IGA3DNurbsXYZ.ForEach(m, test, data);
function test(ig3x, extra)
{
// ig3x is IGA3DNurbsXYZ object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of IGA3DNurbsXYZ objects or properties for all of the IGA 3D Nurbs XYZs in a model in PRIMER. If the optional property argument is not given then an array of IGA3DNurbsXYZ objects is returned. If the property argument is given, that property value for each IGA 3D Nurbs XYZ is returned in the array instead of a IGA3DNurbsXYZ object

Arguments

- **Model** (Model)

Model to get IGA 3D Nurbs XYZs from

- **property (optional)** (string)

Name for property to get for all IGA 3D Nurbs XYZs in the model

Returns

Array of IGA3DNurbsXYZ objects or properties

Return type

Array

Example

To make an array of IGA3DNurbsXYZ objects for all of the IGA 3D Nurbs XYZs in model m:

```
var a = IGA3DNurbsXYZ.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each IGA 3D Nurbs XYZ in model m:

```
var a = IGA3DNurbsXYZ.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a IGA 3D Nurbs XYZ.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the IGA 3D Nurbs XYZ ig3x:

```
var comm_array = ig3x.GetComments();
```

GetControlPoint(index*[integer]*)

Description

Returns the data for a specific control point an an array. There are nr x ns x nt control points.

Arguments

- **index** (integer)

Index you want the control data for. **Note that indices start at 0**.

Returns

An array containing the control point data.

Return type

Array

Example

To get the data for the 3rd control point in IGA 3d nurbs xyz ig3x:

```
var data = ig3x.GetControlPoint(2);
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of IGA3DNurbsXYZ objects for all of the flagged IGA 3D Nurbs XYZs in a model in PRIMER If the optional property argument is not given then an array of IGA3DNurbsXYZ objects is returned. If the property argument is given, then that property value for each IGA 3D Nurbs XYZ is returned in the array instead of a IGA3DNurbsXYZ object

Arguments

- **Model** (Model)

Model to get IGA 3D Nurbs XYZs from

- **flag** (Flag)

Flag set on the IGA 3D Nurbs XYZs that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged IGA 3D Nurbs XYZs in the model

Returns

Array of IGA3DNurbsXYZ objects or properties

Return type

Array

Example

To make an array of IGA3DNurbsXYZ objects for all of the IGA 3D Nurbs XYZs in model m flagged with f:

```
var ig3x = IGA3DNurbsXYZ.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the IGA 3D Nurbs XYZs in model m flagged with f:

```
var a = IGA3DNurbsXYZ.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the IGA3DNurbsXYZ object for a IGA 3D Nurbs XYZ ID.

Arguments

- **Model** (Model)

Model to find the IGA 3D Nurbs XYZ in

- **number** (integer)

number of the IGA 3D Nurbs XYZ you want the IGA3DNurbsXYZ object for

Returns

IGA3DNurbsXYZ object (or null if IGA 3D Nurbs XYZ does not exist).

Return type

IGA3DNurbsXYZ

Example

To get the IGA3DNurbsXYZ object for IGA 3D Nurbs XYZ 100 in model m

```
var ig3x = IGA3DNurbsXYZ.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a IGA3DNurbsXYZ property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the IGA3DNurbsXYZ.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

IGA 3D Nurbs XYZ property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if IGA3DNurbsXYZ property ig3x.example is a parameter:

```
Options.property_parameter_names = true;
if (ig3x.GetParameter(ig3x.example) ) do_something...
Options.property_parameter_names = false;
```

To check if IGA3DNurbsXYZ property ig3x.example is a parameter by using the GetParameter method:

```
if (ig3x.ViewParameters().GetParameter(ig3x.example) ) do_something...
```

GetRKnotVector(index*[integer]*)

Description

The data for Knot values in the local r-direction.

Arguments

- **index** (integer)

Index you want the r-knot value for. **Note that indices start at 0**.

Returns

Knot values in the local r-direction.

Return type

Number

Example

To get the 3rd r-knot value in IGA 3d nurbs xyz ig3x:

```
var data = ig3x.GetRKnotVector(2);
```

GetSKnotVector(index*[integer]*)

Description

The data for Knot values in the local s-direction.

Arguments

- **index** (integer)

Index you want the s-knot value for. **Note that indices start at 0**.

Returns

Knot values in the local s-direction.

Return type

Number

Example

To get the 3rd s-knot value in IGA 3d nurbs xyz ig3x:

```
var data = ig3x.GetSKnotVector(2);
```

GetTKnotVector(index*[integer]*)

Description

The data for Knot values in the local t-direction.

Arguments

- **index** (integer)

Index you want the t-knot value for. **Note that indices start at 0**.

Returns

Knot values in the local t-direction.

Return type

Number

Example

To get the 3rd t-knot value in IGA 3d nurbs xyz ig3x:

```
var data = ig3x.GetTKnotVector(2);
```

Keyword()

Description

Returns the keyword for this IGA 3d nurbs xyz (*IGA_3D_NURBS_XYZ). **Note that a carriage return is not added**.
See also IGA3DNurbsXYZ.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for IGA 3d nurbs xyz ig3x:

```
var key = ig3x.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the IGA 3d nurbs xyz. **Note that a carriage return is not added**. See also IGA3DNurbsXYZ.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for IGA 3d nurbs xyz ig3x:

```
var cards = ig3x.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last IGA 3D Nurbs XYZ in the model.

Arguments

- **Model** (Model)

Model to get last IGA 3D Nurbs XYZ in

Returns

IGA3DNurbsXYZ object (or null if there are no IGA 3D Nurbs XYZs in the model).

Return type

IGA3DNurbsXYZ

Example

To get the last IGA 3D Nurbs XYZ in model m:

```
var ig3x = IGA3DNurbsXYZ.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free IGA 3D Nurbs XYZ label in the model. Also see IGA3DNurbsXYZ.FirstFreeLabel(), IGA3DNurbsXYZ.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free IGA 3D Nurbs XYZ label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

IGA3DNurbsXYZ label.

Return type

Number

Example

To get the last free IGA 3D Nurbs XYZ label in model m:

```
var label = IGA3DNurbsXYZ.LastFreeLabel(m);
```

Next()

Description

Returns the next IGA 3D Nurbs XYZ in the model.

Arguments

No arguments

Returns

IGA3DNurbsXYZ object (or null if there are no more IGA 3D Nurbs XYZs in the model).

Return type

IGA3DNurbsXYZ

Example

To get the IGA 3D Nurbs XYZ in model m after IGA 3D Nurbs XYZ ig3x:

```
var ig3x = ig3x.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) IGA 3D Nurbs XYZ label in the model. Also see IGA3DNurbsXYZ.FirstFreeLabel(), IGA3DNurbsXYZ.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free IGA 3D Nurbs XYZ label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

IGA3DNurbsXYZ label.

Return type

Number

Example

To get the next free IGA 3D Nurbs XYZ label in model m:

```
var label = IGA3DNurbsXYZ.NextFreeLabel(m);
```

Previous()

Description

Returns the previous IGA 3D Nurbs XYZ in the model.

Arguments

No arguments

Returns

IGA3DNurbsXYZ object (or null if there are no more IGA 3D Nurbs XYZs in the model).

Return type

IGA3DNurbsXYZ

Example

To get the IGA 3D Nurbs XYZ in model m before IGA 3D Nurbs XYZ ig3x:

```
var ig3x = ig3x.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the IGA 3D Nurbs XYZs in the model.

Arguments

- **Model** (Model)

Model that all IGA 3D Nurbs XYZs will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the IGA 3D Nurbs XYZs in model m, from 1000000:

```
IGA3DNurbsXYZ.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged IGA 3D Nurbs XYZs in the model.

Arguments

- **Model** (Model)

Model that all the flagged IGA 3D Nurbs XYZs will be renumbered in

- **flag** (Flag)

Flag set on the IGA 3D Nurbs XYZs that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the IGA 3D Nurbs XYZs in model m flagged with f, from 1000000:

```
IGA3DNurbsXYZ.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select IGA 3D Nurbs XYZs using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting IGA 3D Nurbs XYZs

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only IGA 3D Nurbs XYZs from that model can be selected. If the argument is a Flag then only IGA 3D Nurbs XYZs that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any IGA 3D Nurbs XYZs can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of IGA 3D Nurbs XYZs selected or null if menu cancelled

Return type

Number

Example

To select IGA 3D Nurbs XYZs from model m, flagging those selected with flag f, giving the prompt 'Select IGA 3D Nurbs XYZs':

```
IGA3DNurbsXYZ.Select(f, 'Select IGA 3D Nurbs XYZs', m);
```

To select IGA 3D Nurbs XYZs, flagging those selected with flag f but limiting selection to IGA 3D Nurbs XYZs flagged with flag l, giving the prompt 'Select IGA 3D Nurbs XYZs':

```
IGA3DNurbsXYZ.Select(f, 'Select IGA 3D Nurbs XYZs', l);
```

SetControlPoint(index*[integer]*, data*[Array of data]*)

Description

Set the data for a specific control point. For each control point there will be control point X, Y, Z and control weight WGT. There are nr x ns x nt control points.

Arguments

- **index** (integer)

Index you want the control point data for. **Note that indices start at 0**.

- **data** (Array of data)

Array containing the control point data. The array length should be 4.

Returns

No return value.

Example

To set the 3rd control point data for IGA 3d nurbs xyz ig3x to the values in array data:

```
ig3x.SetControlPoint(2, data);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the IGA 3D Nurbs XYZ.

Arguments

- **flag** (Flag)

Flag to set on the IGA 3D Nurbs XYZ

Returns

No return value

Example

To set flag f for IGA 3D Nurbs XYZ ig3x:

```
ig3x.SetFlag(f);
```

SetRKnotVector(index*[integer]*, r*[real]*)

Description

Sets knot value, r in *IGA_3D_NURBS_XYZ. There are [nr + pr + 1] knot values.

Arguments

- **index** (integer)

Index you want to set the r knot vector for. **Note that indices start at 0**.

- **r** (real)

Knot values in the local r-direction

Returns

No return value.

Example

To set the 3rd knot value for IGA 3d nurbs xyz ig3x as 13.5:

```
ig3x.SetRKnotVector(2, 13.5);
```

SetSKnotVector(index*[integer]*, r*[real]*)

Description

Sets knot value, s in *IGA_3D_NURBS_XYZ. There are [ns + ps + 1] knot values.

Arguments

- **index** (integer)

Index you want to set the s knot vector for. **Note that indices start at 0**.

- **r** (real)

Knot values in the local s-direction

Returns

No return value.

Example

To set the 2rd knot value for IGA 3d nurbs xyz ig3x as 24.5:

```
ig3x.SetSKnotVector(1, 24.5);
```

SetTKnotVector(index*[integer]*, r*[real]*)

Description

Sets knot value, t in *IGA_3D_NURBS_XYZ. There are [nt + pt + 1] knot values.

Arguments

- **index** (integer)

Index you want to set the t knot vector for. **Note that indices start at 0**.

- **r** (real)

Knot values in the local t-direction

Returns

No return value.

Example

To set the 2rd knot value for IGA 3d nurbs xyz ig3x as 24.5:

```
ig3x.SetTKnotVector(1, 24.5);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the IGA 3D Nurbs XYZ. The IGA 3D Nurbs XYZ will be sketched until you either call IGA3DNurbsXYZ.Unsketch(), IGA3DNurbsXYZ.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 3D Nurbs XYZ is sketched. If omitted redraw is true. If you want to sketch several IGA 3D Nurbs XYZs and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch IGA 3D Nurbs XYZ ig3x:

```
ig3x.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged IGA 3D Nurbs XYZs in the model. The IGA 3D Nurbs XYZs will be sketched until you either call IGA3DNurbsXYZ.Unsketch(), IGA3DNurbsXYZ.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged IGA 3D Nurbs XYZs will be sketched in

- **flag** (Flag)

Flag set on the IGA 3D Nurbs XYZs that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 3D Nurbs XYZs are sketched. If omitted redraw is true. If you want to sketch flagged IGA 3D Nurbs XYZs several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all IGA 3D Nurbs XYZs flagged with flag in model m:

```
IGA3DNurbsXYZ.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of IGA 3D Nurbs XYZs in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing IGA 3D Nurbs XYZs should be counted. If false or omitted referenced but undefined IGA 3D Nurbs XYZs will also be included in the total.

Returns

number of IGA 3D Nurbs XYZs

Return type

Number

Example

To get the total number of IGA 3D Nurbs XYZs in model m:

```
var total = IGA3DNurbsXYZ.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the IGA 3D Nurbs XYZs in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all IGA 3D Nurbs XYZs will be unset in

- **flag** (Flag)

Flag to unset on the IGA 3D Nurbs XYZs

Returns

No return value

Example

To unset the flag f on all the IGA 3D Nurbs XYZs in model m:

```
IGA3DNurbsXYZ.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the IGA 3D Nurbs XYZ.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 3D Nurbs XYZ is unsketched. If omitted redraw is true. If you want to unsketch several IGA 3D Nurbs XYZs and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch IGA 3D Nurbs XYZ ig3x:

```
ig3x.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all IGA 3D Nurbs XYZs.

Arguments

- **Model** (Model)

Model that all IGA 3D Nurbs XYZs will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 3D Nurbs XYZs are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all IGA 3D Nurbs XYZs in model m:

```
IGA3DNurbsXYZ.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged IGA 3D Nurbs XYZs in the model.

Arguments

- **Model** (Model)

Model that all IGA 3D Nurbs XYZs will be unsketched in

- **flag** (Flag)

Flag set on the IGA 3D Nurbs XYZs that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA 3D Nurbs XYZs are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all IGA 3D Nurbs XYZs flagged with flag in model m:

```
IGA3DNurbsXYZ.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

IGA3DNurbsXYZ object.

Return type

IGA3DNurbsXYZ

Example

To check if IGA3DNurbsXYZ property ig3x.example is a parameter by using the IGA3DNurbsXYZ.GetParameter() method:

```
if (ig3x.ViewParameters().GetParameter(ig3x.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for IGA 3D Nurbs XYZ with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA 3D Nurbs XYZ object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for IGA 3D Nurbs XYZ ig3x:

```
ig3x.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for IGA 3D Nurbs XYZ ig3x:

```
ig3x.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for IGA 3D Nurbs XYZ. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for IGA 3D Nurbs XYZ ig3x:

```
ig3x.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this IGA 3D Nurbs XYZ.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for IGA 3D Nurbs XYZ ig3x:

```
var xrefs = ig3x.Xrefs();
```

---

toString()

Description

Creates a string containing the IGA 3d nurbs xyz data in keyword format. Note that this contains the keyword header and the keyword cards. See also IGA3DNurbsXYZ.Keyword() and IGA3DNurbsXYZ.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for IGA 3d nurbs xyz ig3x in keyword format

```
var s = ig3x.toString();
```

IGAEdgeUVW class

The IGAEdgeUVW class gives you access to IGA edge uvw cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]

- Xrefs()
- toString()

IGAEdgeUVW constants

Name	Description
IGAEdgeUVW.BASIS_TRANSFORM	BASIS_TRANSFORM is *IGA_EDGE_UVW_BASIS_TRANSFORM.
IGAEdgeUVW.NONE	NONE is *IGA_EDGE_UVW.

IGAEdgeUVW properties

Name	Type	Description
edgeid	integer	Parametric Edge ID.
eid	integer	IGAEdgeUVW number. Also see the label property which is an alternative name for this.
elid	integer	Element ID local to a basis transform.
exists (read only)	logical	true if IGA Edge UVW exists, false if referred to but not defined.
exyzid	integer	Physical edge ID. *IGA_EDGE_XYZ.
include	integer	The Include file number that the IGA Edge UVW is in.
label	integer	IGAEdgeUVW number. Also see the eid property which is an alternative name for this.
model (read only)	integer	The Model number that the IGA Edge UVW is in.
option	constant	IGA Edge UVW option. • IGAEdgeUVW.NONE or • IGAEdgeUVW.BASIS_TRANSFORM
patchid	integer	Parametric univariate NURBS patch ID.
rend	real	Parametric coordinate defining the end of the trimmed parametric NURBS.
rstart	real	Parametric coordinate defining the start of the trimmed parametric NURBS.
sense	integer	Sense of orientation with respect to the physical edge.

Detailed Description

The IGAEdgeUVW class allows you to create, modify, edit and manipulate IGA edge uvw cards. See the documentation below for more details.

Constructor

new IGAEdgeUVW(Model*[Model]*, details*[object]*)

Description

Create a new IGAEdgeUVW object.

Arguments

- **Model** (Model)

Model that IGA edge uvw will be created in

- **details** (object)

Details for creating the IGAEdgeUVW

Object has the following properties:

Name	Type	Description
edgeid (optional)	integer	Parametric Edge ID.
eid	integer	ID of the IGA Edge UVW.
elid (optional)	integer	Element ID local to a basis transform.
exyzid (optional)	integer	Physical edge ID. *IGA_EDGE_XYZ.
option (optional)	constant	IGA Edge UVW option. IGAEdgeUVW.NONE or IGAEdgeUVW.BASIS_TRANSFORM
patchid (optional)	integer	Parametric univariate NURBS patch ID.
rend (optional)	real	Parametric coordinate defining the end of the trimmed parametric NURBS.
rstart (optional)	real	Parametric coordinate defining the start of the trimmed parametric NURBS.
sense (optional)	integer	Sense of orientation with respect to the physical edge.

Returns

IGAEdgeUVW object

Return type

IGAEdgeUVW

Example

To create a new IGA edge uvw in model m using eid 10:

```
var igeu = new IGAEdgeUVW(m, {eid: 10});
```

To create a new IGA edge uvw in model m using eid 21 and option BASIS_TRANSFORM with exyzid 22, elid 31 and edgeid as 32:

```
var igeu = new IGAEdgeUVW(m, {option:IGAEdgeUVW.BASIS_TRANSFORM, eid: 21,
exyzid: 22, elid: 31, edgeid: 32});
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a IGA Edge UVW.

Arguments

- **Comment** (Comment)

Comment that will be attached to the IGA Edge UVW

Returns

No return value

Example

To associate comment c to the IGA Edge UVW igeu:

```
igeu.AssociateComment(c);
```

Blank()

Description

Blanks the IGA Edge UVW

Arguments

No arguments

Returns

No return value

Example

To blank IGA Edge UVW igeu:

```
igeu.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the IGA Edge UVWs in the model.

Arguments

- **Model** (Model)

Model that all IGA Edge UVWs will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the IGA Edge UVWs in model m:

```
IGAEdgeUVW.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged IGA Edge UVWs in the model.

Arguments

- **Model** (Model)

Model that all the flagged IGA Edge UVWs will be blanked in

- **flag** (Flag)

Flag set on the IGA Edge UVWs that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the IGA Edge UVWs in model m flagged with f:

```
IGAEdgeUVW.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the IGA Edge UVW is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if IGA Edge UVW igeu is blanked:

```
if (igeu.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse IGA Edge UVW igeu:

```
igeu.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the IGA Edge UVW.

Arguments

- **flag** (Flag)

Flag to clear on the IGA Edge UVW

Returns

No return value

Example

To clear flag f for IGA Edge UVW igeu:

```
igeu.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the IGA Edge UVW. The target include of the copied IGA Edge UVW can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

IGAEdgeUVW object

Return type

IGAEdgeUVW

Example

To copy IGA Edge UVW igeu into IGA Edge UVW z:

```
var z = igeu.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a IGA Edge UVW

Arguments

- **Model** (Model)

Model that the IGA Edge UVW will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

IGAEdgeUVW object (or null if not made).

Return type

IGAEdgeUVW

Example

To start creating a IGA Edge UVW in model m:

```
var igeu = IGAEdgeUVW.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a IGA Edge UVW.

Arguments

- **Comment** (Comment)

Comment that will be detached from the IGA Edge UVW

Returns

No return value

Example

To detach comment c from the IGA Edge UVW igeu:

```
igeu.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit IGA Edge UVW igeu:

```
igeu.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for IGA Edge UVW with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA Edge UVW object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for IGA Edge UVW igeu:

```
igeu.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for IGA Edge UVW igeu:

```
igeu.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for IGA Edge UVW. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for IGA Edge UVW igeu:

```
igeu.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first IGA Edge UVW in the model.

Arguments

- **Model** (Model)

Model to get first IGA Edge UVW in

Returns

IGAEdgeUVW object (or null if there are no IGA Edge UVWs in the model).

Return type

IGAEdgeUVW

Example

To get the first IGA Edge UVW in model m:

```
var igeu = IGAEdgeUVW.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free IGA Edge UVW label in the model. Also see IGAEdgeUVW.LastFreeLabel(), IGAEdgeUVW.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free IGA Edge UVW label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

IGAEdgeUVW label.

Return type

Number

Example

To get the first free IGA Edge UVW label in model m:

```
var label = IGAEdgeUVW.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the IGA Edge UVWs in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all IGA Edge UVWs will be flagged in

- **flag** (Flag)

Flag to set on the IGA Edge UVWs

Returns

No return value

Example

To flag all of the IGA Edge UVWs with flag f in model m:

```
IGAEdgeUVW.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the IGA Edge UVW is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the IGA Edge UVW

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if IGA Edge UVW igeu has flag f set on it:

```
if (igeu.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each IGA Edge UVW in the model.
Note that ForEach has been designed to make looping over IGA Edge UVWs as fast as possible and so has some limitations.
Firstly, a single temporary IGAEdgeUVW object is created and on each function call it is updated with the current IGA Edge UVW data. This means that you should not try to store the IGAEdgeUVW object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new IGA Edge UVWs inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all IGA Edge UVWs are in

- **func** (function)

Function to call for each IGA Edge UVW

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the IGA Edge UVWs in model m:

```
IGAEdgeUVW.ForEach(m, test);
function test(igeu)
{
// igeu is IGAEdgeUVW object
}
```

To call function test for all of the IGA Edge UVWs in model m with optional object:

```
var data = { x:0, y:0 };
IGAEdgeUVW.ForEach(m, test, data);
function test(igeu, extra)
{
// igeu is IGAEdgeUVW object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of IGAEdgeUVW objects or properties for all of the IGA Edge UVWs in a model in PRIMER. If the optional property argument is not given then an array of IGAEdgeUVW objects is returned. If the property argument is given, that property value for each IGA Edge UVW is returned in the array instead of a IGAEdgeUVW object

Arguments

- **Model** (Model)

Model to get IGA Edge UVWs from

- **property (optional)** (string)

Name for property to get for all IGA Edge UVWs in the model

Returns

Array of IGAEdgeUVW objects or properties

Return type

Array

Example

To make an array of IGAEdgeUVW objects for all of the IGA Edge UVWs in model m:

```
var a = IGAEdgeUVW.GetAll(m);
```

To return an array containing the value of property ’foo’ (for example ’x’ for a node) for each IGA Edge UVW in model m:

```
var a = IGAEdgeUVW.GetAll(m, ’foo’);
```

GetComments()

Description

Extracts the comments associated to a IGA Edge UVW.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the IGA Edge UVW igeu:

```
var comm_array = igeu.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of IGAEdgeUVW objects for all of the flagged IGA Edge UVWs in a model in PRIMER If the optional property argument is not given then an array of IGAEdgeUVW objects is returned. If the property argument is given, then that property value for each IGA Edge UVW is returned in the array instead of a IGAEdgeUVW object

Arguments

- **Model** (Model)

Model to get IGA Edge UVWs from

- **flag** (Flag)

Flag set on the IGA Edge UVWs that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged IGA Edge UVWs in the model

Returns

Array of IGAEdgeUVW objects or properties

Return type

Array

Example

To make an array of IGAEdgeUVW objects for all of the IGA Edge UVWs in model m flagged with f:

```
var igeu = IGAEdgeUVW.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the IGA Edge UVWs in model m flagged with f:

```
var a = IGAEdgeUVW.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the IGAEdgeUVW object for a IGA Edge UVW ID.

Arguments

- **Model** (Model)

Model to find the IGA Edge UVW in

- **number** (integer)

number of the IGA Edge UVW you want the IGAEdgeUVW object for

Returns

IGAEdgeUVW object (or null if IGA Edge UVW does not exist).

Return type

IGAEdgeUVW

Example

To get the IGAEdgeUVW object for IGA Edge UVW 100 in model m

```
var igeu = IGAEdgeUVW.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a IGAEdgeUVW property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the IGAEdgeUVW.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

IGA Edge UVW property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if IGAEdgeUVW property igeu.example is a parameter:

```
Options.property_parameter_names = true;
if (igeu.GetParameter(igeu.example) ) do_something...
Options.property_parameter_names = false;
```

To check if IGAEdgeUVW property igeu.example is a parameter by using the GetParameter method:

```
if (igeu.ViewParameters().GetParameter(igeu.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this IGA edge uvw (*IGA_EDGE_UVW). **Note that a carriage return is not added**. See also IGAEdgeUVW.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for IGA edge uvw igeu:

```
var key = igeu.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the IGA edge uvw. **Note that a carriage return is not added**. See also IGAEdgeUVW.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for IGA edge uvw igeu:

```
var cards = igeu.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last IGA Edge UVW in the model.

Arguments

- **Model** (Model)

Model to get last IGA Edge UVW in

Returns

IGAEdgeUVW object (or null if there are no IGA Edge UVWs in the model).

Return type

IGAEdgeUVW

Example

To get the last IGA Edge UVW in model m:

```
var igeu = IGAEdgeUVW.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free IGA Edge UVW label in the model. Also see IGAEdgeUVW.FirstFreeLabel(), IGAEdgeUVW.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free IGA Edge UVW label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

IGAEdgeUVW label.

Return type

Number

Example

To get the last free IGA Edge UVW label in model m:

```
var label = IGAEdgeUVW.LastFreeLabel(m);
```

Next()

Description

Returns the next IGA Edge UVW in the model.

Arguments

No arguments

Returns

IGAEdgeUVW object (or null if there are no more IGA Edge UVWs in the model).

Return type

IGAEdgeUVW

Example

To get the IGA Edge UVW in model m after IGA Edge UVW igeu:

```
var igeu = igeu.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) IGA Edge UVW label in the model. Also see IGAEdgeUVW.FirstFreeLabel(), IGAEdgeUVW.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free IGA Edge UVW label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

IGAEdgeUVW label.

Return type

Number

Example

To get the next free IGA Edge UVW label in model m:

```
var label = IGAEdgeUVW.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a IGA Edge UVW.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only IGA Edge UVWs from that model can be picked. If the argument is a Flag then only IGA Edge UVWs that are flagged with *limit* can be selected. If omitted, or null, any IGA Edge UVWs from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

IGAEdgeUVW object (or null if not picked)

Return type

IGAEdgeUVW

Example

To pick a IGA Edge UVW from model m giving the prompt 'Pick IGA Edge UVW from screen':

```
var igeu = IGAEdgeUVW.Pick('Pick IGA Edge UVW from screen', m);
```

Previous()

Description

Returns the previous IGA Edge UVW in the model.

Arguments

No arguments

Returns

IGAEdgeUVW object (or null if there are no more IGA Edge UVWs in the model).

Return type

IGAEdgeUVW

Example

To get the IGA Edge UVW in model m before IGA Edge UVW igeu:

```
var igeu = igeu.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the IGA Edge UVWs in the model.

Arguments

- **Model** (Model)

Model that all IGA Edge UVWs will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the IGA Edge UVWs in model m, from 1000000:

```
IGAEdgeUVW.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged IGA Edge UVWs in the model.

Arguments

- **Model** (Model)

Model that all the flagged IGA Edge UVWs will be renumbered in

- **flag** (Flag)

Flag set on the IGA Edge UVWs that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the IGA Edge UVWs in model m flagged with f, from 1000000:

```
IGAEdgeUVW.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select IGA Edge UVWs using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting IGA Edge UVWs

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only IGA Edge UVWs from that model can be selected. If the argument is a Flag then only IGA Edge UVWs that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any IGA Edge UVWs can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of IGA Edge UVWs selected or null if menu cancelled

Return type

Number

Example

To select IGA Edge UVWs from model m, flagging those selected with flag f, giving the prompt 'Select IGA Edge UVWs':

```
IGAEdgeUVW.Select(f, 'Select IGA Edge UVWs', m);
```

To select IGA Edge UVWs, flagging those selected with flag f but limiting selection to IGA Edge UVWs flagged with flag l, giving the prompt 'Select IGA Edge UVWs':

```
IGAEdgeUVW.Select(f, 'Select IGA Edge UVWs', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the IGA Edge UVW.

Arguments

- **flag** (Flag)

Flag to set on the IGA Edge UVW

Returns

No return value

Example

To set flag f for IGA Edge UVW igeu:

```
igeu.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the IGA Edge UVW. The IGA Edge UVW will be sketched until you either call IGAEdgeUVW.Unsketch(), IGAEdgeUVW.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Edge UVW is sketched. If omitted redraw is true. If you want to sketch several IGA Edge UVWs and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch IGA Edge UVW igeu:

```
igeu.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged IGA Edge UVWs in the model. The IGA Edge UVWs will be sketched until you either call IGAEdgeUVW.Unsketch(), IGAEdgeUVW.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged IGA Edge UVWs will be sketched in

- **flag** (Flag)

Flag set on the IGA Edge UVWs that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Edge UVWs are sketched. If omitted redraw is true. If you want to sketch flagged IGA Edge UVWs several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all IGA Edge UVWs flagged with flag in model m:

```
IGAEdgeUVW.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of IGA Edge UVWs in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing IGA Edge UVWs should be counted. If false or omitted referenced but undefined IGA Edge UVWs will also be included in the total.

Returns

number of IGA Edge UVWs

Return type

Number

Example

To get the total number of IGA Edge UVWs in model m:

```
var total = IGAEdgeUVW.Total(m);
```

Unblank()

Description

Unblanks the IGA Edge UVW

Arguments

No arguments

Returns

No return value

Example

To unblank IGA Edge UVW igeu:

```
igeu.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the IGA Edge UVWs in the model.

Arguments

- **Model** (Model)

Model that all IGA Edge UVWs will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the IGA Edge UVWs in model m:

```
IGAEdgeUVW.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged IGA Edge UVWs in the model.

Arguments

- **Model** (Model)

Model that the flagged IGA Edge UVWs will be unblanked in

- **flag** (Flag)

Flag set on the IGA Edge UVWs that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the IGA Edge UVWs in model m flagged with f:

```
IGAEdgeUVW.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the IGA Edge UVWs in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all IGA Edge UVWs will be unset in

- **flag** (Flag)

Flag to unset on the IGA Edge UVWs

Returns

No return value

Example

To unset the flag f on all the IGA Edge UVWs in model m:

```
IGAEdgeUVW.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the IGA Edge UVW.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Edge UVW is unsketched. If omitted redraw is true. If you want to unsketch several IGA Edge UVWs and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch IGA Edge UVW igeu:

```
igeu.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all IGA Edge UVWs.

Arguments

- **Model** (Model)

Model that all IGA Edge UVWs will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Edge UVWs are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all IGA Edge UVWs in model m:

```
IGAEdgeUVW.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged IGA Edge UVWs in the model.

Arguments

- **Model** (Model)

Model that all IGA Edge UVWs will be unsketched in

- **flag** (Flag)

Flag set on the IGA Edge UVWs that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Edge UVWs are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all IGA Edge UVWs flagged with flag in model m:

```
IGAEdgeUVW.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

IGAEdgeUVW object.

Return type

IGAEdgeUVW

Example

To check if IGAEdgeUVW property igeu.example is a parameter by using the IGAEdgeUVW.GetParameter() method:

```
if (igeu.ViewParameters().GetParameter(igeu.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for IGA Edge UVW with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA Edge UVW object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...

autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for IGA Edge UVW igeu:

```
igeu.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for IGA Edge UVW igeu:

```
igeu.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for IGA Edge UVW. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for IGA Edge UVW igeu:

```
igeu.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this IGA Edge UVW.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for IGA Edge UVW igeu:

```
var xrefs = igeu.Xrefs();
```

---

toString()

Description

Creates a string containing the IGA edge uvw data in keyword format. Note that this contains the keyword header and the keyword cards. See also IGAEdgeUVW.Keyword() and IGAEdgeUVW.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for IGA edge uvw igeu in keyword format

```
var s = igeu.toString();
```

IGAEdgeXYZ class

The IGAEdgeXYZ class gives you access to IGA edge xyz cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

IGAEdgeXYZ properties

Name	Type	Description

eid	integer	IGAEdgeXYZ number. Also see the label property which is an alternative name for this.
exists (read only)	logical	true if IGA Edge XYZ exists, false if referred to but not defined.
include	integer	The Include file number that the IGA Edge XYZ is in.
label	integer	IGAEdgeXYZ number. Also see the eid property which is an alternative name for this.
model (read only)	integer	The Model number that the IGA Edge XYZ is in.
ori	integer	Orientation with respect to the physical univariate NURBS.
patchid	integer	Physical univariate NURBS patch ID.
pidend	integer	Parametric point ID defining the end of the trimmed physical NURBS.
pidstart	integer	Parametric point ID defining the start of the trimmed physical NURBS.
psid	integer	Parametric point set ID. *SET_IGA_POINT_UVW.

Detailed Description

The IGAEdgeXYZ class allows you to create, modify, edit and manipulate IGA edge xyz cards. See the documentation below for more details.

Constructor

new IGAEdgeXYZ(Model*[Model]*, details*[object]*)

Description

Create a new IGAEdgeXYZ object.

Arguments

- **Model** (Model)

Model that IGA edge xyz will be created in

- **details** (object)

Details for creating the IGAEdgeXYZ

Object has the following properties:

Name	Type	Description
eid	integer	ID of the IGA Edge XYZ.
ori (optional)	integer	Orientation with respect to the physical univariate NURBS.
patchid (optional)	integer	Physical univariate NURBS patch ID.
pidend (optional)	integer	Parametric point ID defining the end of the trimmed physical NURBS.
pidstart (optional)	integer	Parametric point ID defining the start of the trimmed physical NURBS.
psid (optional)	integer	Parametric point set ID. *SET_IGA_POINT_UVW.

Returns

IGAEdgeXYZ object

Return type

IGAEdgeXYZ

Example

To create a new IGA edge xyz in model m using eid 10:

```
var igex = new IGAEdgeXYZ(m, {eid: 10});
```

To create a new IGA edge xyz in model m using eid 21 with patch id 22, ori as 23, pidstart as 24, pidend as 25 and psid as 26:

```
var igex = new IGAEdgeXYZ(m, {eid: 21, patchid: 22, ori: 23, pidstart: 24,
pidend: 25, psid: 26});
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a IGA Edge XYZ.

Arguments

- **Comment** (Comment)

Comment that will be attached to the IGA Edge XYZ

Returns

No return value

Example

To associate comment c to the IGA Edge XYZ igex:

```
igex.AssociateComment(c);
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse IGA Edge XYZ igex:

```
igex.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the IGA Edge XYZ.

Arguments

- **flag** (Flag)

Flag to clear on the IGA Edge XYZ

Returns

No return value

Example

To clear flag f for IGA Edge XYZ igex:

```
igex.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the IGA Edge XYZ. The target include of the copied IGA Edge XYZ can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

IGAEdgeXYZ object

Return type

IGAEdgeXYZ

Example

To copy IGA Edge XYZ igex into IGA Edge XYZ z:

```
var z = igex.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a IGA Edge XYZ

Arguments

- **Model** (Model)

Model that the IGA Edge XYZ will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

IGAEdgeXYZ object (or null if not made).

Return type

IGAEdgeXYZ

Example

To start creating a IGA Edge XYZ in model m:

```
var igex = IGAEdgeXYZ.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a IGA Edge XYZ.

Arguments

- **Comment** (Comment)

Comment that will be detached from the IGA Edge XYZ

Returns

No return value

Example

To detach comment c from the IGA Edge XYZ igex:

```
igex.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit IGA Edge XYZ igex:

```
igex.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for IGA Edge XYZ with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA Edge XYZ object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for IGA Edge XYZ igex:

```
igex.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for IGA Edge XYZ igex:

```
igex.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for IGA Edge XYZ. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for IGA Edge XYZ igex:

```
igex.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first IGA Edge XYZ in the model.

Arguments

- **Model** (Model)

Model to get first IGA Edge XYZ in

Returns

IGAEdgeXYZ object (or null if there are no IGA Edge XYZs in the model).

Return type

IGAEdgeXYZ

Example

To get the first IGA Edge XYZ in model m:

```
var igex = IGAEdgeXYZ.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free IGA Edge XYZ label in the model. Also see IGAEdgeXYZ.LastFreeLabel(), IGAEdgeXYZ.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free IGA Edge XYZ label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

IGAEdgeXYZ label.

Return type

Number

Example

To get the first free IGA Edge XYZ label in model m:

```
var label = IGAEdgeXYZ.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the IGA Edge XYZs in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all IGA Edge XYZs will be flagged in

- **flag** (Flag)

Flag to set on the IGA Edge XYZs

Returns

No return value

Example

To flag all of the IGA Edge XYZs with flag f in model m:

```
IGAEdgeXYZ.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the IGA Edge XYZ is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the IGA Edge XYZ

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if IGA Edge XYZ igex has flag f set on it:

```
if (igex.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each IGA Edge XYZ in the model.
Note that ForEach has been designed to make looping over IGA Edge XYZs as fast as possible and so has some limitations.
Firstly, a single temporary IGAEdgeXYZ object is created and on each function call it is updated with the current IGA Edge XYZ data. This means that you should not try to store the IGAEdgeXYZ object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new IGA Edge XYZs inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all IGA Edge XYZs are in

- **func** (function)

Function to call for each IGA Edge XYZ

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the IGA Edge XYZs in model m:

```
IGAEdgeXYZ.ForEach(m, test);
function test(igex)
{
// igex is IGAEdgeXYZ object
}
```

To call function test for all of the IGA Edge XYZs in model m with optional object:

```
var data = { x:0, y:0 };
IGAEdgeXYZ.ForEach(m, test, data);
function test(igex, extra)
{
// igex is IGAEdgeXYZ object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of IGAEdgeXYZ objects or properties for all of the IGA Edge XYZs in a model in PRIMER. If the optional property argument is not given then an array of IGAEdgeXYZ objects is returned. If the property argument is given, that property value for each IGA Edge XYZ is returned in the array instead of a IGAEdgeXYZ object

Arguments

- **Model** (Model)

Model to get IGA Edge XYZs from

- **property (optional)** (string)

Name for property to get for all IGA Edge XYZs in the model

Returns

Array of IGAEdgeXYZ objects or properties

Return type

Array

Example

To make an array of IGAEdgeXYZ objects for all of the IGA Edge XYZs in model m:

```
var a = IGAEdgeXYZ.GetAll(m);
```

To return an array containing the value of property ’foo’ (for example ’x’ for a node) for each IGA Edge XYZ in model m:

```
var a = IGAEdgeXYZ.GetAll(m, ’foo’);
```

GetComments()

Description

Extracts the comments associated to a IGA Edge XYZ.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the IGA Edge XYZ igex:

```
var comm_array = igex.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of IGAEdgeXYZ objects for all of the flagged IGA Edge XYZs in a model in PRIMER If the optional property argument is not given then an array of IGAEdgeXYZ objects is returned. If the property argument is given, then that property value for each IGA Edge XYZ is returned in the array instead of a IGAEdgeXYZ object

Arguments

- **Model** (Model)

Model to get IGA Edge XYZs from

- **flag** (Flag)

Flag set on the IGA Edge XYZs that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged IGA Edge XYZs in the model

Returns

Array of IGAEdgeXYZ objects or properties

Return type

Array

Example

To make an array of IGAEdgeXYZ objects for all of the IGA Edge XYZs in model m flagged with f:

```
var igex = IGAEdgeXYZ.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the IGA Edge XYZs in model m flagged with f:

```
var a = IGAEdgeXYZ.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the IGAEdgeXYZ object for a IGA Edge XYZ ID.

Arguments

- **Model** (Model)

Model to find the IGA Edge XYZ in

- **number** (integer)

number of the IGA Edge XYZ you want the IGAEdgeXYZ object for

Returns

IGAEdgeXYZ object (or null if IGA Edge XYZ does not exist).

Return type

IGAEdgeXYZ

Example

To get the IGAEdgeXYZ object for IGA Edge XYZ 100 in model m

```
var igex = IGAEdgeXYZ.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a IGAEdgeXYZ property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the IGAEdgeXYZ.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

IGA Edge XYZ property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if IGAEdgeXYZ property igex.example is a parameter:

```
Options.property_parameter_names = true;
if (igex.GetParameter(igex.example) ) do_something...
Options.property_parameter_names = false;
```

To check if IGAEdgeXYZ property igex.example is a parameter by using the GetParameter method:

```
if (igex.ViewParameters().GetParameter(igex.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this IGA edge xyz (*IGA_EDGE_XYZ). **Note that a carriage return is not added**. See also IGAEdgeXYZ.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for IGA edge xyz igex:

```
var key = igex.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the IGA edge xyz. **Note that a carriage return is not added**. See also IGAEdgeXYZ.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for IGA edge xyz igex:

```
var cards = igex.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last IGA Edge XYZ in the model.

Arguments

- **Model** (Model)

Model to get last IGA Edge XYZ in

Returns

IGAEdgeXYZ object (or null if there are no IGA Edge XYZs in the model).

Return type

IGAEdgeXYZ

Example

To get the last IGA Edge XYZ in model m:

```
var igex = IGAEdgeXYZ.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free IGA Edge XYZ label in the model. Also see IGAEdgeXYZ.FirstFreeLabel(), IGAEdgeXYZ.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free IGA Edge XYZ label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

IGAEdgeXYZ label.

Return type

Number

Example

To get the last free IGA Edge XYZ label in model m:

```
var label = IGAEdgeXYZ.LastFreeLabel(m);
```

Next()

Description

Returns the next IGA Edge XYZ in the model.

Arguments

No arguments

Returns

IGAEdgeXYZ object (or null if there are no more IGA Edge XYZs in the model).

Return type

IGAEdgeXYZ

Example

To get the IGA Edge XYZ in model m after IGA Edge XYZ igex:

```
var igex = igex.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) IGA Edge XYZ label in the model. Also see IGAEdgeXYZ.FirstFreeLabel(), IGAEdgeXYZ.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free IGA Edge XYZ label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

IGAEdgeXYZ label.

Return type

Number

Example

To get the next free IGA Edge XYZ label in model m:

```
var label = IGAEdgeXYZ.NextFreeLabel(m);
```

Previous()

Description

Returns the previous IGA Edge XYZ in the model.

Arguments

No arguments

Returns

IGAEdgeXYZ object (or null if there are no more IGA Edge XYZs in the model).

Return type

IGAEdgeXYZ

Example

To get the IGA Edge XYZ in model m before IGA Edge XYZ igex:

```
var igex = igex.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the IGA Edge XYZs in the model.

Arguments

- **Model** (Model)

Model that all IGA Edge XYZs will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the IGA Edge XYZs in model m, from 1000000:

```
IGAEdgeXYZ.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged IGA Edge XYZs in the model.

Arguments

- **Model** (Model)

Model that all the flagged IGA Edge XYZs will be renumbered in

- **flag** (Flag)

Flag set on the IGA Edge XYZs that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the IGA Edge XYZs in model m flagged with f, from 1000000:

```
IGAEdgeXYZ.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select IGA Edge XYZs using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting IGA Edge XYZs

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only IGA Edge XYZs from that model can be selected. If the argument is a Flag then only IGA Edge XYZs that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any IGA Edge XYZs can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of IGA Edge XYZs selected or null if menu cancelled

Return type

Number

Example

To select IGA Edge XYZs from model m, flagging those selected with flag f, giving the prompt 'Select IGA Edge XYZs':

```
IGAEdgeXYZ.Select(f, 'Select IGA Edge XYZs', m);
```

To select IGA Edge XYZs, flagging those selected with flag f but limiting selection to IGA Edge XYZs flagged with flag l, giving the prompt 'Select IGA Edge XYZs':

```
IGAEdgeXYZ.Select(f, 'Select IGA Edge XYZs', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the IGA Edge XYZ.

Arguments

- **flag** (Flag)

Flag to set on the IGA Edge XYZ

Returns

No return value

Example

To set flag f for IGA Edge XYZ igex:

```
igex.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the IGA Edge XYZ. The IGA Edge XYZ will be sketched until you either call IGAEdgeXYZ.Unsketch(), IGAEdgeXYZ.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Edge XYZ is sketched. If omitted redraw is true. If you want to sketch several IGA Edge XYZs and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch IGA Edge XYZ igex:

```
igex.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged IGA Edge XYZs in the model. The IGA Edge XYZs will be sketched until you either call IGAEdgeXYZ.Unsketch(), IGAEdgeXYZ.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged IGA Edge XYZs will be sketched in

- **flag** (Flag)

Flag set on the IGA Edge XYZs that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Edge XYZs are sketched. If omitted redraw is true. If you want to sketch flagged IGA Edge XYZs several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all IGA Edge XYZs flagged with flag in model m:

```
IGAEdgeXYZ.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of IGA Edge XYZs in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing IGA Edge XYZs should be counted. If false or omitted referenced but undefined IGA Edge XYZs will also be included in the total.

Returns

number of IGA Edge XYZs

Return type

Number

Example

To get the total number of IGA Edge XYZs in model m:

```
var total = IGAEdgeXYZ.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the IGA Edge XYZs in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all IGA Edge XYZs will be unset in

- **flag** (Flag)

Flag to unset on the IGA Edge XYZs

Returns

No return value

Example

To unset the flag f on all the IGA Edge XYZs in model m:

```
IGAEdgeXYZ.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the IGA Edge XYZ.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Edge XYZ is unsketched. If omitted redraw is true. If you want to unsketch several IGA Edge XYZs and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch IGA Edge XYZ igex:

```
igex.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all IGA Edge XYZs.

Arguments

- **Model** (Model)

Model that all IGA Edge XYZs will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Edge XYZs are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all IGA Edge XYZs in model m:

```
IGAEdgeXYZ.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged IGA Edge XYZs in the model.

Arguments

- **Model** (Model)

Model that all IGA Edge XYZs will be unsketched in

- **flag** (Flag)

Flag set on the IGA Edge XYZs that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Edge XYZs are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all IGA Edge XYZs flagged with flag in model m:

```
IGAEdgeXYZ.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

IGAEdgeXYZ object.

Return type

IGAEdgeXYZ

Example

To check if IGAEdgeXYZ property igex.example is a parameter by using the IGAEdgeXYZ.GetParameter() method:

```
if (igex.ViewParameters().GetParameter(igex.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for IGA Edge XYZ with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA Edge XYZ object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for IGA Edge XYZ igex:

```
igex.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for IGA Edge XYZ igex:

```
igex.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for IGA Edge XYZ. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for IGA Edge XYZ igex:

```
igex.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this IGA Edge XYZ.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for IGA Edge XYZ igex:

```
var xrefs = igex.Xrefs();
```

---

toString()

Description

Creates a string containing the IGA edge xyz data in keyword format. Note that this contains the keyword header and the keyword cards. See also IGAEdgeXYZ.Keyword() and IGAEdgeXYZ.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for IGA edge xyz igex in keyword format

```
var s = igex.toString();
```

IGAFaceUVW class

The IGAFaceUVW class gives you access to IGA face uvw cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetData(index*[integer]*)
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- RemoveData(index*[integer]*)
- SetData(index*[integer]*, brid*[integer]*)
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)

- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

IGAFaceUVW constants

Name	Description
IGAFaceUVW.BASIS_TRANSFORM	BASIS_TRANSFORM is *IGA_FACE_UVW_BASIS_TRANSFORM.
IGAFaceUVW.NONE	NONE is *IGA_FACE_UVW.

IGAFaceUVW properties

Name	Type	Description
elid	integer	Element ID local to a trivariate basis transform. Only for the BASIS_TRANSFORM keyword option.
entries (read only)	integer	Number of Boundary representation IDs (brid) or Element IDs (elid) added in IGA_FACE_UVW. Can be used as index to add a new entry.
exists (read only)	logical	true if IGA Face UVW exists, false if referred to but not defined.
faceid	integer	Face ID. Only for the BASIS_TRANSFORM keyword option.
fid	integer	IGAFaceUVW number. Also see the label property which is an alternative name for this.
fxyzid	integer	Physical face ID. *IGA_FACE_XYZ.
include	integer	The Include file number that the IGA Face UVW is in.
label	integer	IGAFaceUVW number. Also see the fid property which is an alternative name for this.
model (read only)	integer	The Model number that the IGA Face UVW is in.
option	constant	IGA Face UVW option. • IGAFaceUVW.NONE or • IGAFaceUVW.BASIS_TRANSFORM
patchid	integer	Physical bivariate NURBS patch ID or a bivariate Basis Transform patch ID. Ignored for the BASIS_TRANSFORM keyword option.
sense	integer	Sense of orientation with respect to the physical face. Ignored for the BASIS_TRANSFORM keyword option.

Detailed Description

The IGAFaceUVW class allows you to create, modify, edit and manipulate IGA face uvw cards. See the documentation below for more details.

Constructor

new IGAFaceUVW(Model*[Model]*, details*[object]*)

Description

Create a new IGAFaceUVW object.

Arguments

- **Model** (Model)

Model that IGA face uvw will be created in

- **details** (object)

Details for creating the IGAFaceUVW

Object has the following properties:

Name	Type	Description
elid (optional)	integer	Element ID local to a trivariate basis transform.
faceid (optional)	integer	Face ID.
fid	integer	ID of the IGA Face UVW.
fxyzid (optional)	integer	Physical face ID. *IGA_FACE_XYZ.
option	constant	IGA Face UVW option. IGAFaceUVW.NONE or IGAFaceUVW.BASIS_TRANSFORM
patchid (optional)	integer	Physical bivariate NURBS patch ID.
sense (optional)	integer	Sense of orientation with respect to the physical face.

Returns

IGAFaceUVW object

Return type

IGAFaceUVW

Example

To create a new IGA face uvw in model m using fid 10 and option NONE with fxyzid 11, patchid 12 and sense 1:

```
var igfu = new IGAFaceUVW(m, {option:IGAFaceUVW.NONE, fid:10, fxyzid:11,
patchid:12, sense:1});
```

To create a new IGA face uvw in model m using fid 21 and option BASIS_TRANSFORM with elid 31, faceid 6:

```
var igfu = new IGAFaceUVW(m, {option:IGAFaceUVW.BASIS_TRANSFORM, fid:21, elid:
31, faceid:6});
```

Details of functions

AssociateComment(Comment[*Comment*])

Description

Associates a comment with a IGA Face UVW.

Arguments

- **Comment** (Comment)

Comment that will be attached to the IGA Face UVW

Returns

No return value

Example

To associate comment c to the IGA Face UVW igfu:

```
igfu.AssociateComment(c);
```

Blank()

Description

Blanks the IGA Face UVW

Arguments

No arguments

Returns

No return value

Example

To blank IGA Face UVW igfu:

```
igfu.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the IGA Face UVWs in the model.

Arguments

- **Model** (Model)

Model that all IGA Face UVWs will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the IGA Face UVWs in model m:

```
IGAFaceUVW.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged IGA Face UVWs in the model.

Arguments

- **Model** (Model)

Model that all the flagged IGA Face UVWs will be blanked in

- **flag** (Flag)

Flag set on the IGA Face UVWs that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the IGA Face UVWs in model m flagged with f:

```
IGAFaceUVW.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the IGA Face UVW is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if IGA Face UVW igfu is blanked:

```
if (igfu.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse IGA Face UVW igfu:

```
igfu.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the IGA Face UVW.

Arguments

- **flag** (Flag)

Flag to clear on the IGA Face UVW

Returns

No return value

Example

To clear flag f for IGA Face UVW igfu:

```
igfu.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the IGA Face UVW. The target include of the copied IGA Face UVW can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

IGAFaceUVW object

Return type

IGAFaceUVW

Example

To copy IGA Face UVW igfu into IGA Face UVW z:

```
var z = igfu.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a IGA Face UVW

Arguments

- **Model** (Model)

Model that the IGA Face UVW will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

IGAFaceUVW object (or null if not made).

Return type

IGAFaceUVW

Example

To start creating a IGA Face UVW in model m:

```
var igfu = IGAFaceUVW.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a IGA Face UVW.

Arguments

- **Comment** (Comment)

Comment that will be detached from the IGA Face UVW

Returns

No return value

Example

To detach comment c from the IGA Face UVW igfu:

```
igfu.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit IGA Face UVW igfu:

```
igfu.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for IGA Face UVW with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA Face UVW object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for IGA Face UVW igfu:

```
igfu.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for IGA Face UVW igfu:

```
igfu.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for IGA Face UVW. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for IGA Face UVW igfu:

```
igfu.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first IGA Face UVW in the model.

Arguments

- **Model** (Model)

Model to get first IGA Face UVW in

Returns

IGAFaceUVW object (or null if there are no IGA Face UVWs in the model).

Return type

IGAFaceUVW

Example

To get the first IGA Face UVW in model m:

```
var igfu = IGAFaceUVW.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free IGA Face UVW label in the model. Also see IGAFaceUVW.LastFreeLabel(), IGAFaceUVW.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free IGA Face UVW label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

IGAFaceUVW label.

Return type

Number

Example

To get the first free IGA Face UVW label in model m:

```
var label = IGAFaceUVW.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the IGA Face UVWs in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all IGA Face UVWs will be flagged in

- **flag** (Flag)

Flag to set on the IGA Face UVWs

Returns

No return value

Example

To flag all of the IGA Face UVWs with flag f in model m:

```
IGAFaceUVW.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the IGA Face UVW is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the IGA Face UVW

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if IGA Face UVW igfu has flag f set on it:

```
if (igfu.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each IGA Face UVW in the model.
Note that ForEach has been designed to make looping over IGA Face UVWs as fast as possible and so has some limitations.
Firstly, a single temporary IGAFaceUVW object is created and on each function call it is updated with the current IGA Face UVW data. This means that you should not try to store the IGAFaceUVW object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new IGA Face UVWs inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all IGA Face UVWs are in

- **func** (function)

Function to call for each IGA Face UVW

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the IGA Face UVWs in model m:

```
IGAFaceUVW.ForEach(m, test);
function test(igfu)
{
// igfu is IGAFaceUVW object
}
```

To call function test for all of the IGA Face UVWs in model m with optional object:

```
var data = { x:0, y:0 };
IGAFaceUVW.ForEach(m, test, data);
function test(igfu, extra)
{
// igfu is IGAFaceUVW object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of IGAFaceUVW objects or properties for all of the IGA Face UVWs in a model in PRIMER. If the optional property argument is not given then an array of IGAFaceUVW objects is returned. If the property argument is given, that property value for each IGA Face UVW is returned in the array instead of a IGAFaceUVW object

Arguments

- **Model** (Model)

Model to get IGA Face UVWs from

- **property (optional)** (string)

Name for property to get for all IGA Face UVWs in the model

Returns

Array of IGAFaceUVW objects or properties

Return type

Array

Example

To make an array of IGAFaceUVW objects for all of the IGA Face UVWs in model m:

```
var a = IGAFaceUVW.GetAll(m);
```

To return an array containing the value of property ’foo’ (for example ’x’ for a node) for each IGA Face UVW in model m:

```
var a = IGAFaceUVW.GetAll(m, ’foo’);
```

GetComments()

Description

Extracts the comments associated to a IGA Face UVW.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the IGA Face UVW igfu:

```
var comm_array = igfu.GetComments();
```

GetData(index*[integer]*)

Description

Returns the data for brid in *IGA_FACE_UVW. Only valid for option IGAFaceUVW.NONE.

Arguments

- **index** (integer)

Index you want the data for. **Note that indices start at 0**.

Returns

The ID of boundary representation or basis transform element depending on option.

Return type

Number

Example

To get the 3rd data entry in IGA face uvw igfu:

```
var data = igfu.GetData(2);
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of IGAFaceUVW objects for all of the flagged IGA Face UVWs in a model in PRIMER If the optional property argument is not given then an array of IGAFaceUVW objects is returned. If the property argument is given, then that property value for each IGA Face UVW is returned in the array instead of a IGAFaceUVW object

Arguments

- **Model** (Model)

Model to get IGA Face UVWs from

- **flag** (Flag)

Flag set on the IGA Face UVWs that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged IGA Face UVWs in the model

Returns

Array of IGAFaceUVW objects or properties

Return type

Array

Example

To make an array of IGAFaceUVW objects for all of the IGA Face UVWs in model m flagged with f:

```
var igfu = IGAFaceUVW.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the IGA Face UVWs in model m flagged with f:

```
var a = IGAFaceUVW.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the IGAFaceUVW object for a IGA Face UVW ID.

Arguments

- **Model** (Model)

Model to find the IGA Face UVW in

- **number** (integer)

number of the IGA Face UVW you want the IGAFaceUVW object for

Returns

IGAFaceUVW object (or null if IGA Face UVW does not exist).

Return type

IGAFaceUVW

Example

To get the IGAFaceUVW object for IGA Face UVW 100 in model m

```
var igfu = IGAFaceUVW.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a IGAFaceUVW property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the IGAFaceUVW.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

IGA Face UVW property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if IGAFaceUVW property igfu.example is a parameter:

```
Options.property_parameter_names = true;
if (igfu.GetParameter(igfu.example) ) do_something...
Options.property_parameter_names = false;
```

To check if IGAFaceUVW property igfu.example is a parameter by using the GetParameter method:

```
if (igfu.ViewParameters().GetParameter(igfu.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this IGA face uvw (*IGA_FACE_UVW). **Note that a carriage return is not added**. See also IGAFaceUVW.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for IGA face uvw igfu:

```
var key = igfu.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the IGA face uvw. **Note that a carriage return is not added**. See also IGAFaceUVW.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for IGA face uvw igfu:

```
var cards = igfu.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last IGA Face UVW in the model.

Arguments

- **Model** (Model)

Model to get last IGA Face UVW in

Returns

IGAFaceUVW object (or null if there are no IGA Face UVWs in the model).

Return type

IGAFaceUVW

Example

To get the last IGA Face UVW in model m:

```
var igfu = IGAFaceUVW.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free IGA Face UVW label in the model. Also see IGAFaceUVW.FirstFreeLabel(), IGAFaceUVW.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free IGA Face UVW label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

IGAFaceUVW label.

Return type

Number

Example

To get the last free IGA Face UVW label in model m:

```
var label = IGAFaceUVW.LastFreeLabel(m);
```

Next()

Description

Returns the next IGA Face UVW in the model.

Arguments

No arguments

Returns

IGAFaceUVW object (or null if there are no more IGA Face UVWs in the model).

Return type

IGAFaceUVW

Example

To get the IGA Face UVW in model m after IGA Face UVW igfu:

```
var igfu = igfu.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) IGA Face UVW label in the model. Also see IGAFaceUVW.FirstFreeLabel(), IGAFaceUVW.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free IGA Face UVW label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

IGAFaceUVW label.

Return type

Number

Example

To get the next free IGA Face UVW label in model m:

```
var label = IGAFaceUVW.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a IGA Face UVW.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only IGA Face UVWs from that model can be picked. If the argument is a Flag then only IGA Face UVWs that are flagged with *limit* can be selected. If omitted, or null, any IGA Face UVWs from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

IGAFaceUVW object (or null if not picked)

Return type

IGAFaceUVW

Example

To pick a IGA Face UVW from model m giving the prompt 'Pick IGA Face UVW from screen':

```
var igfu = IGAFaceUVW.Pick('Pick IGA Face UVW from screen', m);
```

Previous()

Description

Returns the previous IGA Face UVW in the model.

Arguments

No arguments

Returns

IGAFaceUVW object (or null if there are no more IGA Face UVWs in the model).

Return type

IGAFaceUVW

Example

To get the IGA Face UVW in model m before IGA Face UVW igfu:

```
var igfu = igfu.Previous();
```

RemoveData(index*[integer]*)

Description

Removes brid/elid for an index in *IGA_FACE_UVW. Only valid for option IGAFaceUVW.NONE.

Arguments

- **index** (integer)

The index you want to delete brid/elid for. **Note that indices start at 0, not 1**.

Returns

No return value.

Example

To delete the brid/elid for the 3rd index for *IGA_FACE_UVW in model m:

```
if (igfu.entries >= 3)
{
    igfu.RemoveData(2);
}
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the IGA Face UVWs in the model.

Arguments

- **Model** (Model)

Model that all IGA Face UVWs will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the IGA Face UVWs in model m, from 1000000:

```
IGAFaceUVW.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged IGA Face UVWs in the model.

Arguments

- **Model** (Model)

Model that all the flagged IGA Face UVWs will be renumbered in

- **flag** (Flag)

Flag set on the IGA Face UVWs that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the IGA Face UVWs in model m flagged with f, from 1000000:

```
IGAFaceUVW.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select IGA Face UVWs using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting IGA Face UVWs

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only IGA Face UVWs from that model can be selected. If the argument is a Flag then only IGA Face UVWs that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any IGA Face UVWs can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of IGA Face UVWs selected or null if menu cancelled

Return type

Number

Example

To select IGA Face UVWs from model m, flagging those selected with flag f, giving the prompt 'Select IGA Face UVWs':

```
IGAFaceUVW.Select(f, 'Select IGA Face UVWs', m);
```

To select IGA Face UVWs, flagging those selected with flag f but limiting selection to IGA Face UVWs flagged with flag l, giving the prompt 'Select IGA Face UVWs':

```
IGAFaceUVW.Select(f, 'Select IGA Face UVWs', l);
```

SetData(index*[integer]*, brid*[integer]*)

Description

Sets brid for *IGA_FACE_UVW. Only valid for option IGAFaceUVW.NONE.

Arguments

- **index** (integer)

Index you want to set the brid/elid for. **Note that indices start at 0**.

- **brid** (integer)

The ID of 1 dimensional boundary representation.

Returns

No return value.

Example

To add an entry to the *IGA_FACE_UVW igfu with id 10 by using "entries" property to get the next available index:

```
var i = igfu.entries;
igfu.SetData(i, 10);
```

To change an entry in 3rd field for *IGA_FACE_UVW igfu with id 30:

```
igfu.SetData(2, 30);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the IGA Face UVW.

Arguments

- **flag** (Flag)

Flag to set on the IGA Face UVW

Returns

No return value

Example

To set flag f for IGA Face UVW igfu:

```
igfu.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the IGA Face UVW. The IGA Face UVW will be sketched until you either call IGAFaceUVW.Unsketch(), IGAFaceUVW.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Face UVW is sketched. If omitted redraw is true. If you want to sketch several IGA Face UVWs and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch IGA Face UVW igfu:

```
igfu.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged IGA Face UVWs in the model. The IGA Face UVWs will be sketched until you either call IGAFaceUVW.Unsketch(), IGAFaceUVW.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged IGA Face UVWs will be sketched in

- **flag** (Flag)

Flag set on the IGA Face UVWs that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Face UVWs are sketched. If omitted redraw is true. If you want to sketch flagged IGA Face UVWs several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all IGA Face UVWs flagged with flag in model m:

```
IGAFaceUVW.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of IGA Face UVWs in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing IGA Face UVWs should be counted. If false or omitted referenced but undefined IGA Face UVWs will also be included in the total.

Returns

number of IGA Face UVWs

Return type

Number

Example

To get the total number of IGA Face UVWs in model m:

```
var total = IGAFaceUVW.Total(m);
```

Unblank()

Description

Unblanks the IGA Face UVW

Arguments

No arguments

Returns

No return value

Example

To unblank IGA Face UVW igfu:

```
igfu.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the IGA Face UVWs in the model.

Arguments

- **Model** (Model)

Model that all IGA Face UVWs will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the IGA Face UVWs in model m:

```
IGAFaceUVW.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged IGA Face UVWs in the model.

Arguments

- **Model** (Model)

Model that the flagged IGA Face UVWs will be unblanked in

- **flag** (Flag)

Flag set on the IGA Face UVWs that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the IGA Face UVWs in model m flagged with f:

```
IGAFaceUVW.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the IGA Face UVWs in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all IGA Face UVWs will be unset in

- **flag** (Flag)

Flag to unset on the IGA Face UVWs

Returns

No return value

Example

To unset the flag f on all the IGA Face UVWs in model m:

```
IGAFaceUVW.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the IGA Face UVW.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Face UVW is unsketched. If omitted redraw is true. If you want to unsketch several IGA Face UVWs and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch IGA Face UVW igfu:

```
igfu.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all IGA Face UVWs.

Arguments

- **Model** (Model)

Model that all IGA Face UVWs will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Face UVWs are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all IGA Face UVWs in model m:

```
IGAFaceUVW.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged IGA Face UVWs in the model.

Arguments

- **Model** (Model)

Model that all IGA Face UVWs will be unsketched in

- **flag** (Flag)

Flag set on the IGA Face UVWs that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Face UVWs are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all IGA Face UVWs flagged with flag in model m:

```
IGAFaceUVW.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

IGAFaceUVW object.

Return type

IGAFaceUVW

Example

To check if IGAFaceUVW property igfu.example is a parameter by using the IGAFaceUVW.GetParameter() method:

```
if (igfu.ViewParameters().GetParameter(igfu.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for IGA Face UVW with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description

autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA Face UVW object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for IGA Face UVW igfu:

```
igfu.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for IGA Face UVW igfu:

```
igfu.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for IGA Face UVW. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for IGA Face UVW igfu:

```
igfu.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this IGA Face UVW.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for IGA Face UVW igfu:

```
var xrefs = igfu.Xrefs();
```

---

toString()

Description

Creates a string containing the IGA face uvw data in keyword format. Note that this contains the keyword header and the keyword cards. See also IGAFaceUVW.Keyword() and IGAFaceUVW.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for IGA face uvw igfu in keyword format

```
var s = igfu.toString();
```

IGAFaceXYZ class

The IGAFaceXYZ class gives you access to IGA face xyz cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetData(index*[integer]*)
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- RemoveData(index*[integer]*)
- SetData(index*[integer]*, brid*[integer]*)
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

IGAFaceXYZ constants

Name	Description
IGAFaceXYZ.BASIS_TRANSFORM	BASIS_TRANSFORM is *IGA_FACE_XYZ_BASIS_TRANSFORM.
IGAFaceXYZ.NONE	NONE is *IGA_FACE_XYZ.

IGAFaceXYZ properties

Name	Type	Description
entries (read only)	integer	Number of Boundary representation IDs (brid) or Element IDs (elid) added in IGA_FACE_XYZ. Can be used as index to add a new entry.
esid	integer	Parametric edge set ID. *SET_IGA_EDGE_UVW.
exists (read only)	logical	true if IGA Face XYZ exists, false if referred to but not defined.
fid	integer	IGAFaceXYZ number. Also see the label property which is an alternative name for this.
include	integer	The Include file number that the IGA Face XYZ is in.
label	integer	IGAFaceXYZ number. Also see the fid property which is an alternative name for this.
model (read only)	integer	The Model number that the IGA Face XYZ is in.
option	constant	IGA Face XYZ option. • IGAFaceXYZ.NONE or • IGAFaceXYZ.BASIS_TRANSFORM
ori	integer	Orientation with respect to the physical bivariate NURBS.
patchid	integer	Physical bivariate NURBS patch ID or a bivariate Basis Transform patch ID.
psid	integer	Parametric point set ID. *SET_IGA_POINT_UVW.

Detailed Description

The IGAFaceXYZ class allows you to create, modify, edit and manipulate IGA face xyz cards. See the documentation below for more details.

Constructor

new IGAFaceXYZ(Model*[Model]*, details*[object]*)

Description

Create a new IGAFaceXYZ object.

Arguments

- **Model** (Model)

Model that IGA face xyz will be created in

- **details** (object)

Details for creating the IGAFaceXYZ

Object has the following properties:

Name	Type	Description
esid (optional)	integer	Parametric edge set ID. *SET_IGA_EDGE_UVW.
fid	integer	ID of the IGA Face XYZ.

option	constant	IGA Face XYZ option. IGAFaceXYZ.NONE or IGAFaceXYZ.BASIS_TRANSFORM
ori (optional)	integer	Orientation with respect to the physical bivariate NURBS.
patchid (optional)	integer	Physical bivariate NURBS patch ID or a bivariate basis transform patch ID.
psid (optional)	integer	Parametric point set ID. *SET_IGA_POINT_UVW.

Returns

IGAFaceXYZ object

Return type

IGAFaceXYZ

Example

To create a new IGA face xyz in model m using fid 10 with option NONE:

```
var igfx = new IGAFaceXYZ(m, {option:IGAFaceXYZ.NONE, fid:10});
```

To create a new IGA face xyz in model m using fid 21 and option BASIS_TRANSFORM with patch 22, ori 23, psid 24 and esid 25:

```
var igfx = new IGAFaceXYZ(m, {option:IGAFaceXYZ.BASIS_TRANSFORM, fid:21,
patchid: 22, ori:23, psid:24, esid:25});
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a IGA Face XYZ.

Arguments

- **Comment** (Comment)

Comment that will be attached to the IGA Face XYZ

Returns

No return value

Example

To associate comment c to the IGA Face XYZ igfx:

```
igfx.AssociateComment(c);
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse IGA Face XYZ igfx:

```
igfx.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the IGA Face XYZ.

Arguments

- **flag** (Flag)

Flag to clear on the IGA Face XYZ

Returns

No return value

Example

To clear flag f for IGA Face XYZ igfx:

```
igfx.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the IGA Face XYZ. The target include of the copied IGA Face XYZ can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

IGAFaceXYZ object

Return type

IGAFaceXYZ

Example

To copy IGA Face XYZ igfx into IGA Face XYZ z:

```
var z = igfx.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a IGA Face XYZ

Arguments

- **Model** (Model)

Model that the IGA Face XYZ will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

IGAFaceXYZ object (or null if not made).

Return type

IGAFaceXYZ

Example

To start creating a IGA Face XYZ in model m:

```
var igfx = IGAFaceXYZ.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a IGA Face XYZ.

Arguments

- **Comment** (Comment)

Comment that will be detached from the IGA Face XYZ

Returns

No return value

Example

To detach comment c from the IGA Face XYZ igfx:

```
igfx.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit IGA Face XYZ igfx:

```
igfx.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for IGA Face XYZ with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA Face XYZ object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for IGA Face XYZ igfx:

```
igfx.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for IGA Face XYZ igfx:

```
igfx.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for IGA Face XYZ. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for IGA Face XYZ igfx:

```
igfx.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first IGA Face XYZ in the model.

Arguments

- **Model** (Model)

Model to get first IGA Face XYZ in

Returns

IGAFaceXYZ object (or null if there are no IGA Face XYZs in the model).

Return type

IGAFaceXYZ

Example

To get the first IGA Face XYZ in model m:

```
var igfx = IGAFaceXYZ.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free IGA Face XYZ label in the model. Also see IGAFaceXYZ.LastFreeLabel(), IGAFaceXYZ.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free IGA Face XYZ label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

IGAFaceXYZ label.

Return type

Number

Example

To get the first free IGA Face XYZ label in model m:

```
var label = IGAFaceXYZ.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the IGA Face XYZs in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all IGA Face XYZs will be flagged in

- **flag** (Flag)

Flag to set on the IGA Face XYZs

Returns

No return value

Example

To flag all of the IGA Face XYZs with flag f in model m:

```
IGAFaceXYZ.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the IGA Face XYZ is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the IGA Face XYZ

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if IGA Face XYZ igfx has flag f set on it:

```
if (igfx.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each IGA Face XYZ in the model.
Note that ForEach has been designed to make looping over IGA Face XYZs as fast as possible and so has some limitations.
Firstly, a single temporary IGAFaceXYZ object is created and on each function call it is updated with the current IGA Face XYZ data. This means that you should not try to store the IGAFaceXYZ object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new IGA Face XYZs inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all IGA Face XYZs are in

- **func** (function)

Function to call for each IGA Face XYZ

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the IGA Face XYZs in model m:

```
IGAFaceXYZ.ForEach(m, test);
function test(igfx)
{
// igfx is IGAFaceXYZ object
}
```

To call function test for all of the IGA Face XYZs in model m with optional object:

```
var data = { x:0, y:0 };
IGAFaceXYZ.ForEach(m, test, data);
function test(igfx, extra)
{
// igfx is IGAFaceXYZ object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of IGAFaceXYZ objects or properties for all of the IGA Face XYZs in a model in PRIMER. If the optional property argument is not given then an array of IGAFaceXYZ objects is returned. If the property argument is given, that property value for each IGA Face XYZ is returned in the array instead of a IGAFaceXYZ object

Arguments

- **Model** (Model)

Model to get IGA Face XYZs from

- **property (optional)** (string)

Name for property to get for all IGA Face XYZs in the model

Returns

Array of IGAFaceXYZ objects or properties

Return type

Array

Example

To make an array of IGAFaceXYZ objects for all of the IGA Face XYZs in model m:

```
var a = IGAFaceXYZ.GetAll(m);
```

To return an array containing the value of property ’foo’ (for example ’x’ for a node) for each IGA Face XYZ in model m:

```
var a = IGAFaceXYZ.GetAll(m, ’foo’);
```

GetComments()

Description

Extracts the comments associated to a IGA Face XYZ.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the IGA Face XYZ igfx:

```
var comm_array = igfx.GetComments();
```

GetData(index*[integer]*)

Description

Returns the data for brid/elid based on option in *IGA_FACE_XYZ.

Arguments

- **index** (integer)

Index you want the data for. **Note that indices start at 0**.

Returns

The ID of boundary representation or basis transform element depending on option.

Return type

Number

Example

To get the 3rd data entry in IGA face xyz igfx:

```
var data = igfx.GetData(2);
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of IGAFaceXYZ objects for all of the flagged IGA Face XYZs in a model in PRIMER If the optional property argument is not given then an array of IGAFaceXYZ objects is returned. If the property argument is given, then that property value for each IGA Face XYZ is returned in the array instead of a IGAFaceXYZ object

Arguments

- **Model** (Model)

Model to get IGA Face XYZs from

- **flag** (Flag)

Flag set on the IGA Face XYZs that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged IGA Face XYZs in the model

Returns

Array of IGAFaceXYZ objects or properties

Return type

Array

Example

To make an array of IGAFaceXYZ objects for all of the IGA Face XYZs in model m flagged with f:

```
var igfx = IGAFaceXYZ.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the IGA Face XYZs in model m flagged with f:

```
var a = IGAFaceXYZ.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the IGAFaceXYZ object for a IGA Face XYZ ID.

Arguments

- **Model** (Model)

Model to find the IGA Face XYZ in

- **number** (integer)

number of the IGA Face XYZ you want the IGAFaceXYZ object for

Returns

IGAFaceXYZ object (or null if IGA Face XYZ does not exist).

Return type

IGAFaceXYZ

Example

To get the IGAFaceXYZ object for IGA Face XYZ 100 in model m

```
var igfx = IGAFaceXYZ.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a IGAFaceXYZ property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the IGAFaceXYZ.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

IGA Face XYZ property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if IGAFaceXYZ property igfx.example is a parameter:

```
Options.property_parameter_names = true;
if (igfx.GetParameter(igfx.example) ) do_something...
Options.property_parameter_names = false;
```

To check if IGAFaceXYZ property igfx.example is a parameter by using the GetParameter method:

```
if (igfx.ViewParameters().GetParameter(igfx.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this IGA face xyz (*IGA_FACE_XYZ). **Note that a carriage return is not added**. See also IGAFaceXYZ.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for IGA face xyz igfx:

```
var key = igfx.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the IGA face xyz. **Note that a carriage return is not added**. See also IGAFaceXYZ.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for IGA face xyz igfx:

```
var cards = igfx.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last IGA Face XYZ in the model.

Arguments

- **Model** (Model)

Model to get last IGA Face XYZ in

Returns

IGAFaceXYZ object (or null if there are no IGA Face XYZs in the model).

Return type

IGAFaceXYZ

Example

To get the last IGA Face XYZ in model m:

```
var igfx = IGAFaceXYZ.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free IGA Face XYZ label in the model. Also see IGAFaceXYZ.FirstFreeLabel(), IGAFaceXYZ.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free IGA Face XYZ label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

IGAFaceXYZ label.

Return type

Number

Example

To get the last free IGA Face XYZ label in model m:

```
var label = IGAFaceXYZ.LastFreeLabel(m);
```

Next()

Description

Returns the next IGA Face XYZ in the model.

Arguments

No arguments

Returns

IGAFaceXYZ object (or null if there are no more IGA Face XYZs in the model).

Return type

IGAFaceXYZ

Example

To get the IGA Face XYZ in model m after IGA Face XYZ igfx:

```
var igfx = igfx.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) IGA Face XYZ label in the model. Also see IGAFaceXYZ.FirstFreeLabel(), IGAFaceXYZ.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free IGA Face XYZ label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

IGAFaceXYZ label.

Return type

Number

Example

To get the next free IGA Face XYZ label in model m:

```
var label = IGAFaceXYZ.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a IGA Face XYZ.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only IGA Face XYZs from that model can be picked. If the argument is a Flag then only IGA Face XYZs that are flagged with *limit* can be selected. If omitted, or null, any IGA Face XYZs from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

IGAFaceXYZ object (or null if not picked)

Return type

IGAFaceXYZ

Example

To pick a IGA Face XYZ from model m giving the prompt 'Pick IGA Face XYZ from screen':

```
var igfx = IGAFaceXYZ.Pick('Pick IGA Face XYZ from screen', m);
```

Previous()

Description

Returns the previous IGA Face XYZ in the model.

Arguments

No arguments

Returns

IGAFaceXYZ object (or null if there are no more IGA Face XYZs in the model).

Return type

IGAFaceXYZ

Example

To get the IGA Face XYZ in model m before IGA Face XYZ igfx:

```
var igfx = igfx.Previous();
```

RemoveData(index*[integer]*)

Description

Removes brid/elid for an index in *IGA_FACE_XYZ.

Arguments

- **index** (integer)

The index you want to delete brid/elid for. **Note that indices start at 0, not 1**.

Returns

No return value.

Example

To delete the brid/elid for the 3rd index for *IGA_FACE_XYZ in model m:

```
if (igfx.entries >= 3)
{
    igfx.RemoveData(2);
}
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the IGA Face XYZs in the model.

Arguments

- **Model** (Model)

Model that all IGA Face XYZs will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the IGA Face XYZs in model m, from 1000000:

```
IGAFaceXYZ.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged IGA Face XYZs in the model.

Arguments

- **Model** (Model)

Model that all the flagged IGA Face XYZs will be renumbered in

- **flag** (Flag)

Flag set on the IGA Face XYZs that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the IGA Face XYZs in model m flagged with f, from 1000000:

```
IGAFaceXYZ.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select IGA Face XYZs using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting IGA Face XYZs

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only IGA Face XYZs from that model can be selected. If the argument is a Flag then only IGA Face XYZs that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any IGA Face XYZs can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of IGA Face XYZs selected or null if menu cancelled

Return type

Number

Example

To select IGA Face XYZs from model m, flagging those selected with flag f, giving the prompt 'Select IGA Face XYZs':

```
IGAFaceXYZ.Select(f, 'Select IGA Face XYZs', m);
```

To select IGA Face XYZs, flagging those selected with flag f but limiting selection to IGA Face XYZs flagged with flag l, giving the prompt 'Select IGA Face XYZs':

```
IGAFaceXYZ.Select(f, 'Select IGA Face XYZs', l);
```

SetData(index*[integer]*, brid*[integer]*)

Description

Sets brid/elid based on option for an *IGA_FACE_XYZ.

Arguments

- **index** (integer)

Index you want to set the brid/elid for. **Note that indices start at 0**.

- **brid** (integer)

The ID of boundary representation or basis transform element depending on option.

Returns

No return value.

Example

To add an entry to the *IGA_FACE_XYZ igfx with id 10 by using "entries" property to get the next available index:

```
var i = igfx.entries;
igfx.SetData(i, 10);
```

To change an entry in 3rd field for *IGA_FACE_XYZ igfx with id 30:

```
igfx.SetData(2, 30);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the IGA Face XYZ.

Arguments

- **flag** (Flag)

Flag to set on the IGA Face XYZ

Returns

No return value

Example

To set flag f for IGA Face XYZ igfx:

```
igfx.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the IGA Face XYZ. The IGA Face XYZ will be sketched until you either call IGAFaceXYZ.Unsketch(), IGAFaceXYZ.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Face XYZ is sketched. If omitted redraw is true. If you want to sketch several IGA Face XYZs and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch IGA Face XYZ igfx:

```
igfx.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged IGA Face XYZs in the model. The IGA Face XYZs will be sketched until you either call IGAFaceXYZ.Unsketch(), IGAFaceXYZ.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged IGA Face XYZs will be sketched in

- **flag** (Flag)

Flag set on the IGA Face XYZs that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Face XYZs are sketched. If omitted redraw is true. If you want to sketch flagged IGA Face XYZs several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all IGA Face XYZs flagged with flag in model m:

```
IGAFaceXYZ.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of IGA Face XYZs in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing IGA Face XYZs should be counted. If false or omitted referenced but undefined IGA Face XYZs will also be included in the total.

Returns

number of IGA Face XYZs

Return type

Number

Example

To get the total number of IGA Face XYZs in model m:

```
var total = IGAFaceXYZ.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the IGA Face XYZs in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all IGA Face XYZs will be unset in

- **flag** (Flag)

Flag to unset on the IGA Face XYZs

Returns

No return value

Example

To unset the flag f on all the IGA Face XYZs in model m:

```
IGAFaceXYZ.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the IGA Face XYZ.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Face XYZ is unsketched. If omitted redraw is true. If you want to unsketch several IGA Face XYZs and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch IGA Face XYZ igfx:

```
igfx.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all IGA Face XYZs.

Arguments

- **Model** (Model)

Model that all IGA Face XYZs will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Face XYZs are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all IGA Face XYZs in model m:

```
IGAFaceXYZ.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged IGA Face XYZs in the model.

Arguments

- **Model** (Model)

Model that all IGA Face XYZs will be unsketched in

- **flag** (Flag)

Flag set on the IGA Face XYZs that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Face XYZs are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all IGA Face XYZs flagged with flag in model m:

```
IGAFaceXYZ.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

IGAFaceXYZ object.

Return type

IGAFaceXYZ

Example

To check if IGAFaceXYZ property igfx.example is a parameter by using the IGAFaceXYZ.GetParameter() method:

```
if (igfx.ViewParameters().GetParameter(igfx.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for IGA Face XYZ with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA Face XYZ object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for IGA Face XYZ igfx:

```
igfx.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for IGA Face XYZ igfx:

```
igfx.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for IGA Face XYZ. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for IGA Face XYZ igfx:

```
igfx.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this IGA Face XYZ.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for IGA Face XYZ igfx:

```
var xrefs = igfx.Xrefs();
```

---

toString()

Description

Creates a string containing the IGA face xyz data in keyword format. Note that this contains the keyword header and the keyword cards. See also IGAFaceXYZ.Keyword() and IGAFaceXYZ.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for IGA face xyz igfx in keyword format

```
var s = igfx.toString();
```

IGAIntegrationShellReduce class

The IGAIntegrationShellReduce class gives you access to IGA integration shell reduce cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

IGAIntegrationShellReduce properties

Name	Type	Description
exists (read only)	logical	true if IGA Integration Shell Reduce exists, false if referred to but not defined.
id (read only)	integer	ID of the IGA integration shell reduce. Only used in PRIMER.
include	integer	The Include file number that the IGA Integration Shell Reduce is in.

model (read only)	integer	The Model number that the IGA Integration Shell Reduce is in.
nrdr	int	Reduced degree in the r-direction.
nrds	int	Reduced degree in the s-direction.
patchid	integer	PATCH ID defined in *IGA_SHELL.

Detailed Description

The IGAIntegrationShellReduce class allows you to create, modify, edit and manipulate IGA integration shell reduce cards. See the documentation below for more details.

Constructor

new IGAIntegrationShellReduce(Model*[Model]*, details*[object]*)

Description

Create a new IGAIntegrationShellReduce object.

Arguments

- **Model** (Model)

Model that IGA integration shell reduce will be created in

- **details** (object)

Details for creating the IGAIntegrationShellReduce

Object has the following properties:

Name	Type	Description
nrdr (optional)	int	Reduced degree in the r-direction.
nrds (optional)	int	Reduced degree in the s-direction.
patchid	integer	PATCH ID defined in *IGA_SHELL.

Returns

IGAIntegrationShellReduce object

Return type

IGAIntegrationShellReduce

Example

To create a new IGA integration shell reduce in model m for shell patch 10:

```
var igish = new IGAIntegrationShellReduce(m, {patchid: 10});
```

To create a new IGA integration shell reduce in model m for shell patch 20 with nrdr and nrds 1 and 2 respectively:

```
var igish = new IGAIntegrationShellReduce(m, {patchid: 20, nrdr: 1, nrds: 2});
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a IGA Integration Shell Reduce.

Arguments

- **Comment** (Comment)

Comment that will be attached to the IGA Integration Shell Reduce

Returns

No return value

Example

To associate comment c to the IGA Integration Shell Reduce igish:

```
igish.AssociateComment(c);
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse IGA Integration Shell Reduce igish:

```
igish.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the IGA Integration Shell Reduce.

Arguments

- **flag** (Flag)

Flag to clear on the IGA Integration Shell Reduce

Returns

No return value

Example

To clear flag f for IGA Integration Shell Reduce igish:

```
igish.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the IGA Integration Shell Reduce. The target include of the copied IGA Integration Shell Reduce can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

IGAIntegrationShellReduce object

Return type

IGAIntegrationShellReduce

Example

To copy IGA Integration Shell Reduce igish into IGA Integration Shell Reduce z:

```
var z = igish.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a IGA Integration Shell Reduce

Arguments

- **Model** (Model)

Model that the IGA Integration Shell Reduce will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

IGAIntegrationShellReduce object (or null if not made).

Return type

IGAIntegrationShellReduce

Example

To start creating a IGA Integration Shell Reduce in model m:

```
var igish = IGAIntegrationShellReduce.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a IGA Integration Shell Reduce.

Arguments

- **Comment** (Comment)

Comment that will be detached from the IGA Integration Shell Reduce

Returns

No return value

Example

To detach comment c from the IGA Integration Shell Reduce igish:

```
igish.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit IGA Integration Shell Reduce igish:

```
igish.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for IGA Integration Shell Reduce with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA Integration Shell Reduce object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for IGA Integration Shell Reduce igish:

```
igish.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for IGA Integration Shell Reduce igish:

```
igish.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for IGA Integration Shell Reduce. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for IGA Integration Shell Reduce igish:

```
igish.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first IGA Integration Shell Reduce in the model.

Arguments

- **Model** (Model)

Model to get first IGA Integration Shell Reduce in

Returns

IGAIntegrationShellReduce object (or null if there are no IGA Integration Shell Reduces in the model).

Return type

IGAIntegrationShellReduce

Example

To get the first IGA Integration Shell Reduce in model m:

```
var igish = IGAIntegrationShellReduce.First(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the IGA Integration Shell Reduces in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all IGA Integration Shell Reduces will be flagged in

- **flag** (Flag)

Flag to set on the IGA Integration Shell Reduces

Returns

No return value

Example

To flag all of the IGA Integration Shell Reduces with flag f in model m:

```
IGAIntegrationShellReduce.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the IGA Integration Shell Reduce is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the IGA Integration Shell Reduce

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if IGA Integration Shell Reduce igish has flag f set on it:

```
if (igish.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each IGA Integration Shell Reduce in the model.
Note that ForEach has been designed to make looping over IGA Integration Shell Reduces as fast as possible and so has some limitations.
Firstly, a single temporary IGAIntegrationShellReduce object is created and on each function call it is updated with the current IGA Integration Shell Reduce data. This means that you should not try to store the IGAIntegrationShellReduce object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new IGA Integration Shell Reduces inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all IGA Integration Shell Reduces are in

- **func** (function)

Function to call for each IGA Integration Shell Reduce

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the IGA Integration Shell Reduces in model m:

```
IGAIntegrationShellReduce.ForEach(m, test);
function test(igish)
{
// igish is IGAIntegrationShellReduce object
}
```

To call function test for all of the IGA Integration Shell Reduces in model m with optional object:

```
var data = { x:0, y:0 };
IGAIntegrationShellReduce.ForEach(m, test, data);
function test(igish, extra)
{
// igish is IGAIntegrationShellReduce object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of IGAIntegrationShellReduce objects or properties for all of the IGA Integration Shell Reduces in a model in PRIMER. If the optional property argument is not given then an array of IGAIntegrationShellReduce objects is returned. If the property argument is given, that property value for each IGA Integration Shell Reduce is returned in the array instead of a IGAIntegrationShellReduce object

Arguments

- **Model** (Model)

Model to get IGA Integration Shell Reduces from

- **property (optional)** (string)

Name for property to get for all IGA Integration Shell Reduces in the model

Returns

Array of IGAIntegrationShellReduce objects or properties

Return type

Array

Example

To make an array of IGAIntegrationShellReduce objects for all of the IGA Integration Shell Reduces in model m:

```
var a = IGAIntegrationShellReduce.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each IGA Integration Shell Reduce in model m:

```
var a = IGAIntegrationShellReduce.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a IGA Integration Shell Reduce.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the IGA Integration Shell Reduce igish:

```
var comm_array = igish.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of IGAIntegrationShellReduce objects for all of the flagged IGA Integration Shell Reduces in a model in PRIMER If the optional property argument is not given then an array of IGAIntegrationShellReduce objects is returned. If the property argument is given, then that property value for each IGA Integration Shell Reduce is returned in the array instead of a IGAIntegrationShellReduce object

Arguments

- **Model** (Model)

Model to get IGA Integration Shell Reduces from

- **flag** (Flag)

Flag set on the IGA Integration Shell Reduces that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged IGA Integration Shell Reduces in the model

Returns

Array of IGAIntegrationShellReduce objects or properties

Return type

Array

Example

To make an array of IGAIntegrationShellReduce objects for all of the IGA Integration Shell Reduces in model m flagged with f:

```
var igish = IGAIntegrationShellReduce.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the IGA Integration Shell Reduces in model m flagged with f:

```
var a = IGAIntegrationShellReduce.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the IGAIntegrationShellReduce object for a IGA Integration Shell Reduce ID.

Arguments

- **Model** (Model)

Model to find the IGA Integration Shell Reduce in

- **number** (integer)

number of the IGA Integration Shell Reduce you want the IGAIntegrationShellReduce object for

Returns

IGAIntegrationShellReduce object (or null if IGA Integration Shell Reduce does not exist).

Return type

IGAIntegrationShellReduce

Example

To get the IGAIntegrationShellReduce object for IGA Integration Shell Reduce 100 in model m

```
var igish = IGAIntegrationShellReduce.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a IGAIntegrationShellReduce property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the IGAIntegrationShellReduce.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

IGA Integration Shell Reduce property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if IGAIntegrationShellReduce property igish.example is a parameter:

```
Options.property_parameter_names = true;
if (igish.GetParameter(igish.example) ) do_something...
Options.property_parameter_names = false;
```

To check if IGAIntegrationShellReduce property igish.example is a parameter by using the GetParameter method:

```
if (igish.ViewParameters().GetParameter(igish.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this IGA integration shell reduce (*IGA_SHELL). **Note that a carriage return is not added**. See also IGAIntegrationShellReduce.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for IGA integration shell reduce igs:

```
var key = igs.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the IGA integration shell reduce. **Note that a carriage return is not added**. See also IGAIntegrationShellReduce.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for IGA integration shell reduce igs:

```
var cards = igs.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last IGA Integration Shell Reduce in the model.

Arguments

- **Model** (Model)

Model to get last IGA Integration Shell Reduce in

Returns

IGAIntegrationShellReduce object (or null if there are no IGA Integration Shell Reduces in the model).

Return type

IGAIntegrationShellReduce

Example

To get the last IGA Integration Shell Reduce in model m:

```
var igish = IGAIntegrationShellReduce.Last(m);
```

Next()

Description

Returns the next IGA Integration Shell Reduce in the model.

Arguments

No arguments

Returns

IGAIntegrationShellReduce object (or null if there are no more IGA Integration Shell Reduces in the model).

Return type

IGAIntegrationShellReduce

Example

To get the IGA Integration Shell Reduce in model m after IGA Integration Shell Reduce igish:

```
var igish = igish.Next();
```

Previous()

Description

Returns the previous IGA Integration Shell Reduce in the model.

Arguments

No arguments

Returns

IGAIntegrationShellReduce object (or null if there are no more IGA Integration Shell Reduces in the model).

Return type

IGAIntegrationShellReduce

Example

To get the IGA Integration Shell Reduce in model m before IGA Integration Shell Reduce igish:

```
var igish = igish.Previous();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select IGA Integration Shell Reduces using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting IGA Integration Shell Reduces

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only IGA Integration Shell Reduces from that model can be selected. If the argument is a Flag then only IGA Integration Shell Reduces that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any IGA Integration Shell Reduces can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of IGA Integration Shell Reduces selected or null if menu cancelled

Return type

Number

Example

To select IGA Integration Shell Reduces from model m, flagging those selected with flag f, giving the prompt 'Select IGA Integration Shell Reduces':

```
IGAIntegrationShellReduce.Select(f, 'Select IGA Integration Shell Reduces', m);
```

To select IGA Integration Shell Reduces, flagging those selected with flag f but limiting selection to IGA Integration Shell Reduces flagged with flag l, giving the prompt 'Select IGA Integration Shell Reduces':

```
IGAIntegrationShellReduce.Select(f, 'Select IGA Integration Shell Reduces', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the IGA Integration Shell Reduce.

Arguments

- **flag** (Flag)

Flag to set on the IGA Integration Shell Reduce

Returns

No return value

Example

To set flag f for IGA Integration Shell Reduce igish:

```
igish.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the IGA Integration Shell Reduce. The IGA Integration Shell Reduce will be sketched until you either call IGAIntegrationShellReduce.Unsketch(), IGAIntegrationShellReduce.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Integration Shell Reduce is sketched. If omitted redraw is true. If you want to sketch several IGA Integration Shell Reduces and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch IGA Integration Shell Reduce igish:

```
igish.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged IGA Integration Shell Reduces in the model. The IGA Integration Shell Reduces will be sketched until you either call IGAIntegrationShellReduce.Unsketch(), IGAIntegrationShellReduce.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged IGA Integration Shell Reduces will be sketched in

- **flag** (Flag)

Flag set on the IGA Integration Shell Reduces that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Integration Shell Reduces are sketched. If omitted redraw is true. If you want to sketch flagged IGA Integration Shell Reduces several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all IGA Integration Shell Reduces flagged with flag in model m:

```
IGAIntegrationShellReduce.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of IGA Integration Shell Reduces in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing IGA Integration Shell Reduces should be counted. If false or omitted referenced but undefined IGA Integration Shell Reduces will also be included in the total.

Returns

number of IGA Integration Shell Reduces

Return type

Number

Example

To get the total number of IGA Integration Shell Reduces in model m:

```
var total = IGAIntegrationShellReduce.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the IGA Integration Shell Reduces in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all IGA Integration Shell Reduces will be unset in

- **flag** (Flag)

Flag to unset on the IGA Integration Shell Reduces

Returns

No return value

Example

To unset the flag f on all the IGA Integration Shell Reduces in model m:

```
IGAIntegrationShellReduce.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the IGA Integration Shell Reduce.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Integration Shell Reduce is unsketched. If omitted redraw is true. If you want to unsketch several IGA Integration Shell Reduces and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch IGA Integration Shell Reduce igish:

```
igish.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all IGA Integration Shell Reduces.

Arguments

- **Model** (Model)

Model that all IGA Integration Shell Reduces will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Integration Shell Reduces are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all IGA Integration Shell Reduces in model m:

```
IGAIntegrationShellReduce.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged IGA Integration Shell Reduces in the model.

Arguments

- **Model** (Model)

Model that all IGA Integration Shell Reduces will be unsketched in

- **flag** (Flag)

Flag set on the IGA Integration Shell Reduces that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Integration Shell Reduces are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all IGA Integration Shell Reduces flagged with flag in model m:

```
IGAIntegrationShellReduce.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

IGAIntegrationShellReduce object.

Return type

IGAIntegrationShellReduce

Example

To check if IGAIntegrationShellReduce property igish.example is a parameter by using the IGAIntegrationShellReduce.GetParameter() method:

```
if (igish.ViewParameters().GetParameter(igish.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for IGA Integration Shell Reduce with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA Integration Shell Reduce object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for IGA Integration Shell Reduce igish:

```
igish.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for IGA Integration Shell Reduce igish:

```
igish.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for IGA Integration Shell Reduce. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for IGA Integration Shell Reduce igish:

```
igish.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this IGA Integration Shell Reduce.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for IGA Integration Shell Reduce igish:

```
var xrefs = igish.Xrefs();
```

---

toString()

Description

Creates a string containing the IGA integration shell reduce data in keyword format. Note that this contains the keyword header and the keyword cards. See also IGAIntegrationShellReduce.Keyword() and IGAIntegrationShellReduce.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for IGA integration shell reduce igs in keyword format

```
var s = igs.toString();
```

IGAIntegrationSolidReduce class

The IGAIntegrationSolidReduce class gives you access to IGA integration solid reduce cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

IGAIntegrationSolidReduce properties

Name	Type	Description
exists (read only)	logical	true if IGA Integration Solid Reduce exists, false if referred to but not defined.
id (read only)	integer	ID of the IGA integration solid reduce. Only used in PRIMER.
include	integer	The Include file number that the IGA Integration Solid Reduce is in.

model (read only)	integer	The Model number that the IGA Integration Solid Reduce is in.
nrdr	int	Reduced degree in the r-direction.
nrds	int	Reduced degree in the s-direction.
nrdt	int	Reduced degree in the t-direction.
patchid	integer	PATCH ID defined in *IGA_SOLID.

Detailed Description

The IGAIntegrationSolidReduce class allows you to create, modify, edit and manipulate IGA integration solid reduce cards. See the documentation below for more details.

Constructor

new IGAIntegrationSolidReduce(Model*[Model]*, details*[object]*)

Description

Create a new IGAIntegrationSolidReduce object.

Arguments

- **Model** (Model)

Model that IGA integration solid reduce will be created in

- **details** (object)

Details for creating the IGAIntegrationSolidReduce

Object has the following properties:

Name	Type	Description
nrdr (optional)	int	Reduced degree in the r-direction.
nrds (optional)	int	Reduced degree in the s-direction.
nrdt (optional)	int	Reduced degree in the t-direction.
patchid	integer	PATCH ID defined in *IGA_SOLID.

Returns

IGAIntegrationSolidReduce object

Return type

IGAIntegrationSolidReduce

Example

To create a new IGA integration solid reduce in model m for solid patch 10:

```
var igiso = new IGAIntegrationSolidReduce(m, {patchid: 10});
```

To create a new IGA integration solid reduce in model m for solid patch 20 with nrdr, nrds and nrdt as 1, 2 and 2 respectively:

```
var igiso = new IGAIntegrationSolidReduce(m, {patchid: 20, nrdr: 1, nrds: 2,
nrdt: 2});
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a IGA Integration Solid Reduce.

Arguments

- **Comment** (Comment)

Comment that will be attached to the IGA Integration Solid Reduce

Returns

No return value

Example

To associate comment c to the IGA Integration Solid Reduce igiso:

```
igiso.AssociateComment(c);
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse IGA Integration Solid Reduce igiso:

```
igiso.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the IGA Integration Solid Reduce.

Arguments

- **flag** (Flag)

Flag to clear on the IGA Integration Solid Reduce

Returns

No return value

Example

To clear flag f for IGA Integration Solid Reduce igiso:

```
igiso.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the IGA Integration Solid Reduce. The target include of the copied IGA Integration Solid Reduce can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

IGAIntegrationSolidReduce object

Return type

IGAIntegrationSolidReduce

Example

To copy IGA Integration Solid Reduce igiso into IGA Integration Solid Reduce z:

```
var z = igiso.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a IGA Integration Solid Reduce

Arguments

- **Model** (Model)

Model that the IGA Integration Solid Reduce will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

IGAIntegrationSolidReduce object (or null if not made).

Return type

IGAIntegrationSolidReduce

Example

To start creating a IGA Integration Solid Reduce in model m:

```
var igiso = IGAIntegrationSolidReduce.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a IGA Integration Solid Reduce.

Arguments

- **Comment** (Comment)

Comment that will be detached from the IGA Integration Solid Reduce

Returns

No return value

Example

To detach comment c from the IGA Integration Solid Reduce igiso:

```
igiso.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit IGA Integration Solid Reduce igiso:

```
igiso.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for IGA Integration Solid Reduce with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description

autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA Integration Solid Reduce object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for IGA Integration Solid Reduce igiso:

```
igiso.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for IGA Integration Solid Reduce igiso:

```
igiso.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for IGA Integration Solid Reduce. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for IGA Integration Solid Reduce igiso:

```
igiso.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first IGA Integration Solid Reduce in the model.

Arguments

- **Model** (Model)

Model to get first IGA Integration Solid Reduce in

Returns

IGAIntegrationSolidReduce object (or null if there are no IGA Integration Solid Reduces in the model).

Return type

IGAIntegrationSolidReduce

Example

To get the first IGA Integration Solid Reduce in model m:

```
var igiso = IGAIntegrationSolidReduce.First(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the IGA Integration Solid Reduces in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all IGA Integration Solid Reduces will be flagged in

- **flag** (Flag)

Flag to set on the IGA Integration Solid Reduces

Returns

No return value

Example

To flag all of the IGA Integration Solid Reduces with flag f in model m:

```
IGAIntegrationSolidReduce.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the IGA Integration Solid Reduce is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the IGA Integration Solid Reduce

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if IGA Integration Solid Reduce igiso has flag f set on it:

```
if (igiso.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each IGA Integration Solid Reduce in the model.
Note that ForEach has been designed to make looping over IGA Integration Solid Reduces as fast as possible and so has some limitations.
Firstly, a single temporary IGAIntegrationSolidReduce object is created and on each function call it is updated with the current IGA Integration Solid Reduce data. This means that you should not try to store the IGAIntegrationSolidReduce object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new IGA Integration Solid Reduces inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all IGA Integration Solid Reduces are in

- **func** (function)

Function to call for each IGA Integration Solid Reduce

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the IGA Integration Solid Reduces in model m:

```
IGAIntegrationSolidReduce.ForEach(m, test);
function test(igiso)
{
// igiso is IGAIntegrationSolidReduce object
}
```

To call function test for all of the IGA Integration Solid Reduces in model m with optional object:

```
var data = { x:0, y:0 };
IGAIntegrationSolidReduce.ForEach(m, test, data);
function test(igiso, extra)
{
// igiso is IGAIntegrationSolidReduce object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of IGAIntegrationSolidReduce objects or properties for all of the IGA Integration Solid Reduces in a model in PRIMER. If the optional property argument is not given then an array of IGAIntegrationSolidReduce objects is returned. If the property argument is given, that property value for each IGA Integration Solid Reduce is returned in the array instead of a IGAIntegrationSolidReduce object

Arguments

- **Model** (Model)

Model to get IGA Integration Solid Reduces from

- **property (optional)** (string)

Name for property to get for all IGA Integration Solid Reduces in the model

Returns

Array of IGAIntegrationSolidReduce objects or properties

Return type

Array

Example

To make an array of IGAIntegrationSolidReduce objects for all of the IGA Integration Solid Reduces in model m:

```
var a = IGAIntegrationSolidReduce.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each IGA Integration Solid Reduce in model m:

```
var a = IGAIntegrationSolidReduce.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a IGA Integration Solid Reduce.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the IGA Integration Solid Reduce igiso:

```
var comm_array = igiso.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of IGAIntegrationSolidReduce objects for all of the flagged IGA Integration Solid Reduces in a model in PRIMER If the optional property argument is not given then an array of IGAIntegrationSolidReduce objects is returned. If the property argument is given, then that property value for each IGA Integration Solid Reduce is returned in the array instead of a IGAIntegrationSolidReduce object

Arguments

- **Model** (Model)

Model to get IGA Integration Solid Reduces from

- **flag** (Flag)

Flag set on the IGA Integration Solid Reduces that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged IGA Integration Solid Reduces in the model

Returns

Array of IGAIntegrationSolidReduce objects or properties

Return type

Array

Example

To make an array of IGAIntegrationSolidReduce objects for all of the IGA Integration Solid Reduces in model m flagged with f:

```
var igiso = IGAIntegrationSolidReduce.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the IGA Integration Solid Reduces in model m flagged with f:

```
var a = IGAIntegrationSolidReduce.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the IGAIntegrationSolidReduce object for a IGA Integration Solid Reduce ID.

Arguments

- **Model** (Model)

Model to find the IGA Integration Solid Reduce in

- **number** (integer)

number of the IGA Integration Solid Reduce you want the IGAIntegrationSolidReduce object for

Returns

IGAIntegrationSolidReduce object (or null if IGA Integration Solid Reduce does not exist).

Return type

IGAIntegrationSolidReduce

Example

To get the IGAIntegrationSolidReduce object for IGA Integration Solid Reduce 100 in model m

```
var igiso = IGAIntegrationSolidReduce.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a IGAIntegrationSolidReduce property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the IGAIntegrationSolidReduce.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

IGA Integration Solid Reduce property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if IGAIntegrationSolidReduce property igiso.example is a parameter:

```
Options.property_parameter_names = true;
if (igiso.GetParameter(igiso.example) ) do_something...
Options.property_parameter_names = false;
```

To check if IGAIntegrationSolidReduce property igiso.example is a parameter by using the GetParameter method:

```
if (igiso.ViewParameters().GetParameter(igiso.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this IGA integration solid reduce (*IGA_SOLID). **Note that a carriage return is not added**. See also IGAIntegrationSolidReduce.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for IGA integration solid reduce igs:

```
var key = igs.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the IGA integration solid reduce. **Note that a carriage return is not added**. See also IGAIntegrationSolidReduce.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for IGA integration solid reduce igs:

```
var cards = igs.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last IGA Integration Solid Reduce in the model.

Arguments

- **Model** (Model)

Model to get last IGA Integration Solid Reduce in

Returns

IGAIntegrationSolidReduce object (or null if there are no IGA Integration Solid Reduces in the model).

Return type

IGAIntegrationSolidReduce

Example

To get the last IGA Integration Solid Reduce in model m:

```
var igiso = IGAIntegrationSolidReduce.Last(m);
```

Next()

Description

Returns the next IGA Integration Solid Reduce in the model.

Arguments

No arguments

Returns

IGAIntegrationSolidReduce object (or null if there are no more IGA Integration Solid Reduces in the model).

Return type

IGAIntegrationSolidReduce

Example

To get the IGA Integration Solid Reduce in model m after IGA Integration Solid Reduce igiso:

```
var igiso = igiso.Next();
```

Previous()

Description

Returns the previous IGA Integration Solid Reduce in the model.

Arguments

No arguments

Returns

IGAIntegrationSolidReduce object (or null if there are no more IGA Integration Solid Reduces in the model).

Return type

IGAIntegrationSolidReduce

Example

To get the IGA Integration Solid Reduce in model m before IGA Integration Solid Reduce igiso:

```
var igiso = igiso.Previous();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select IGA Integration Solid Reduces using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting IGA Integration Solid Reduces

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only IGA Integration Solid Reduces from that model can be selected. If the argument is a Flag then only IGA Integration Solid Reduces that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any IGA Integration Solid Reduces can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of IGA Integration Solid Reduces selected or null if menu cancelled

Return type

Number

Example

To select IGA Integration Solid Reduces from model m, flagging those selected with flag f, giving the prompt 'Select IGA Integration Solid Reduces':

```
IGAIntegrationSolidReduce.Select(f, 'Select IGA Integration Solid Reduces', m);
```

To select IGA Integration Solid Reduces, flagging those selected with flag f but limiting selection to IGA Integration Solid Reduces flagged with flag l, giving the prompt 'Select IGA Integration Solid Reduces':

```
IGAIntegrationSolidReduce.Select(f, 'Select IGA Integration Solid Reduces', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the IGA Integration Solid Reduce.

Arguments

- **flag** (Flag)

Flag to set on the IGA Integration Solid Reduce

Returns

No return value

Example

To set flag f for IGA Integration Solid Reduce igiso:

```
igiso.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the IGA Integration Solid Reduce. The IGA Integration Solid Reduce will be sketched until you either call IGAIntegrationSolidReduce.Unsketch(), IGAIntegrationSolidReduce.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Integration Solid Reduce is sketched. If omitted redraw is true. If you want to sketch several IGA Integration Solid Reduces and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch IGA Integration Solid Reduce igiso:

```
igiso.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged IGA Integration Solid Reduces in the model. The IGA Integration Solid Reduces will be sketched until you either call IGAIntegrationSolidReduce.Unsketch(), IGAIntegrationSolidReduce.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged IGA Integration Solid Reduces will be sketched in

- **flag** (Flag)

Flag set on the IGA Integration Solid Reduces that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Integration Solid Reduces are sketched. If omitted redraw is true. If you want to sketch flagged IGA Integration Solid Reduces several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all IGA Integration Solid Reduces flagged with flag in model m:

```
IGAIntegrationSolidReduce.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of IGA Integration Solid Reduces in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing IGA Integration Solid Reduces should be counted. If false or omitted referenced but undefined IGA Integration Solid Reduces will also be included in the total.

Returns

number of IGA Integration Solid Reduces

Return type

Number

Example

To get the total number of IGA Integration Solid Reduces in model m:

```
var total = IGAIntegrationSolidReduce.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the IGA Integration Solid Reduces in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all IGA Integration Solid Reduces will be unset in

- **flag** (Flag)

Flag to unset on the IGA Integration Solid Reduces

Returns

No return value

Example

To unset the flag f on all the IGA Integration Solid Reduces in model m:

```
IGAIntegrationSolidReduce.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the IGA Integration Solid Reduce.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Integration Solid Reduce is unsketched. If omitted redraw is true. If you want to unsketch several IGA Integration Solid Reduces and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch IGA Integration Solid Reduce igiso:

```
igiso.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all IGA Integration Solid Reduces.

Arguments

- **Model** (Model)

Model that all IGA Integration Solid Reduces will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Integration Solid Reduces are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all IGA Integration Solid Reduces in model m:

```
IGAIntegrationSolidReduce.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged IGA Integration Solid Reduces in the model.

Arguments

- **Model** (Model)

Model that all IGA Integration Solid Reduces will be unsketched in

- **flag** (Flag)

Flag set on the IGA Integration Solid Reduces that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Integration Solid Reduces are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all IGA Integration Solid Reduces flagged with flag in model m:

```
IGAIntegrationSolidReduce.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

IGAIntegrationSolidReduce object.

Return type

IGAIntegrationSolidReduce

Example

To check if IGAIntegrationSolidReduce property igiso.example is a parameter by using the IGAIntegrationSolidReduce.GetParameter() method:

```
if (igiso.ViewParameters().GetParameter(igiso.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for IGA Integration Solid Reduce with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description

autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA Integration Solid Reduce object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for IGA Integration Solid Reduce igiso:

```
igiso.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for IGA Integration Solid Reduce igiso:

```
igiso.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for IGA Integration Solid Reduce. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for IGA Integration Solid Reduce igiso:

```
igiso.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this IGA Integration Solid Reduce.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for IGA Integration Solid Reduce igiso:

```
var xrefs = igiso.Xrefs();
```

---

toString()

Description

Creates a string containing the IGA integration solid reduce data in keyword format. Note that this contains the keyword header and the keyword cards. See also IGAIntegrationSolidReduce.Keyword() and IGAIntegrationSolidReduce.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for IGA integration solid reduce igs in keyword format

```
var s = igs.toString();
```

IGAMass class

The IGAMass class gives you access to IGA mass cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

IGAMass constants

Name	Description
IGAMass.EDGE_UVW	EDGE is *IGA_MASS_EDGE_UVW.
IGAMass.FACE_XYZ	FACE is *IGA_MASS_FACE_XYZ.
IGAMass.POINT_UVW	POINT is *IGA_MASS_POINT_UVW.

IGAMass.SET_EDGE_UVW	S_EDGE is *IGA_MASS_SET_EDGE_UVW.
IGAMass.SET_FACE_XYZ	S_FACE is *IGA_MASS_SET_FACE_XYZ.
IGAMass.SET_POINT_UVW	S_POINT is *IGA_MASS_SET_POINT_UVW.

IGAMass properties

Name	Type	Description
dmass	real	Uniformly distributed mass added to IGA entity.
exists (read only)	logical	true if IGA Mass exists, false if referred to but not defined.
id	integer	ID of IGA entity depending on the choice of keyword option. Can be IGA_POINT_UVW, IGA_EDGE_UVW, IGA_FACE_XYZ, SET_IGA_POINT_UVW, SET_IGA_EDGE_UVW or SET_IGA_FACE_XYZ.
include	integer	The Include file number that the IGA Mass is in.
label (read only)	integer	IGAMass number. Only used in PRIMER.
model (read only)	integer	The Model number that the IGA Mass is in.
option	constant	IGA mass option. Can be IGAMass.POINT_UVW, IGAMass.EDGE_UVW, IGAMass.FACE_XYZ, IGAMass.SET_POINT_UVW, IGAMass.SET_EDGE_UVW or IGAMass.SET_FACE_XYZ.
tmass	real	Mass added to IGA entity.

Detailed Description

The IGAMass class allows you to create, modify, edit and manipulate IGA mass cards. See the documentation below for more details.

Constructor

new IGAMass(Model*[Model]*, details*[object]*)

Description

Create a new IGAMass object.

Arguments

- **Model** (Model)

Model that IGA mass will be created in

- **details** (object)

Details for creating the IGAMass

Object has the following properties:

Name	Type	Description
dmass (optional)	real	Uniformly distributed mass added to IGA entity.
id	integer	ID of IGA entity depending on the choice of keyword option. Can be IGA_POINT_UVW, IGA_EDGE_UVW, IGA_FACE_XYZ, SET_IGA_POINT_UVW, SET_IGA_EDGE_UVW or SET_IGA_FACE_XYZ.

option	constant	IGA Face XYZ option. Can be IGAMass.POINT_UVW, IGAMass.EDGE_UVW, IGAMass.FACE_XYZ, IGAMass.SET_POINT_UVW, IGAMass.SET_EDGE_UVW or IGAMass.SET_FACE_XYZ.
tmass (optional)	real	Mass added to IGA entity.

Returns

IGAMass object

Return type

IGAMass

Example

To create a new IGA mass in model m using option POINT with id 10:

```
var igma = new IGAMass(m, {option:IGAMass.POINT_UVW, id:10});
```

To create a new IGA mass in model m using option S_EDGE with id 21, tmass 22.5 and dmass 23.5:

```
var igma = new IGAMass(m, {option:IGAMass.SET_EDGE_UVW, id:21, tmass: 22.5,
dmass:23.5});
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a IGA Mass.

Arguments

- **Comment** (Comment)

Comment that will be attached to the IGA Mass

Returns

No return value

Example

To associate comment c to the IGA Mass igma:

```
igma.AssociateComment(c);
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse IGA Mass igma:

```
igma.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the IGA Mass.

Arguments

- **flag** (Flag)

Flag to clear on the IGA Mass

Returns

No return value

Example

To clear flag f for IGA Mass igma:

```
igma.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the IGA Mass. The target include of the copied IGA Mass can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

IGAMass object

Return type

IGAMass

Example

To copy IGA Mass igma into IGA Mass z:

```
var z = igma.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a IGA Mass

Arguments

- **Model** (Model)

Model that the IGA Mass will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

IGAMass object (or null if not made).

Return type

IGAMass

Example

To start creating a IGA Mass in model m:

```
var igma = IGAMass.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a IGA Mass.

Arguments

- **Comment** (Comment)

Comment that will be detached from the IGA Mass

Returns

No return value

Example

To detach comment c from the IGA Mass igma:

```
igma.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit IGA Mass igma:

```
igma.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for IGA Mass with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA Mass object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for IGA Mass igma:

```
igma.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for IGA Mass igma:

```
igma.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for IGA Mass. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for IGA Mass igma:

```
igma.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first IGA Mass in the model.

Arguments

- **Model** (Model)

Model to get first IGA Mass in

Returns

IGAMass object (or null if there are no IGA Masss in the model).

Return type

IGAMass

Example

To get the first IGA Mass in model m:

```
var igma = IGAMass.First(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the IGA Masss in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all IGA Masss will be flagged in

- **flag** (Flag)

Flag to set on the IGA Masss

Returns

No return value

Example

To flag all of the IGA Masss with flag f in model m:

```
IGAMass.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the IGA Mass is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the IGA Mass

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if IGA Mass igma has flag f set on it:

```
if (igma.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each IGA Mass in the model.
Note that ForEach has been designed to make looping over IGA Masss as fast as possible and so has some limitations.
Firstly, a single temporary IGAMass object is created and on each function call it is updated with the current IGA Mass data. This means that you should not try to store the IGAMass object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new IGA Masss inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all IGA Masss are in

- **func** (function)

Function to call for each IGA Mass

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the IGA Masss in model m:

```
IGAMass.ForEach(m, test);
function test(igma)
{
// igma is IGAMass object
}
```

To call function test for all of the IGA Masss in model m with optional object:

```
var data = { x:0, y:0 };
IGAMass.ForEach(m, test, data);
function test(igma, extra)
{
// igma is IGAMass object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of IGAMass objects or properties for all of the IGA Masss in a model in PRIMER. If the optional property argument is not given then an array of IGAMass objects is returned. If the property argument is given, that property value for each IGA Mass is returned in the array instead of a IGAMass object

Arguments

- **Model** (Model)

Model to get IGA Masss from

- **property (optional)** (string)

Name for property to get for all IGA Masss in the model

Returns

Array of IGAMass objects or properties

Return type

Array

Example

To make an array of IGAMass objects for all of the IGA Masss in model m:

```
var a = IGAMass.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each IGA Mass in model m:

```
var a = IGAMass.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a IGA Mass.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the IGA Mass igma:

```
var comm_array = igma.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of IGAMass objects for all of the flagged IGA Masss in a model in PRIMER If the optional property argument is not given then an array of IGAMass objects is returned. If the property argument is given, then that property value for each IGA Mass is returned in the array instead of a IGAMass object

Arguments

- **Model** (Model)

Model to get IGA Masss from

- **flag** (Flag)

Flag set on the IGA Masss that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged IGA Masss in the model

Returns

Array of IGAMass objects or properties

Return type

Array

Example

To make an array of IGAMass objects for all of the IGA Masss in model m flagged with f:

```
var igma = IGAMass.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the IGA Masss in model m flagged with f:

```
var a = IGAMass.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the IGAMass object for a IGA Mass ID.

Arguments

- **Model** (Model)

Model to find the IGA Mass in

- **number** (integer)

number of the IGA Mass you want the IGAMass object for

Returns

IGAMass object (or null if IGA Mass does not exist).

Return type

IGAMass

Example

To get the IGAMass object for IGA Mass 100 in model m

```
var igma = IGAMass.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a IGAMass property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the IGAMass.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

IGA Mass property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if IGAMass property igma.example is a parameter:

```
Options.property_parameter_names = true;
if (igma.GetParameter(igma.example) ) do_something...
Options.property_parameter_names = false;
```

To check if IGAMass property igma.example is a parameter by using the GetParameter method:

```
if (igma.ViewParameters().GetParameter(igma.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this IGA mass (*IGA_MASS). **Note that a carriage return is not added**. See also IGAMass.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for IGA mass igma:

```
var key = igma.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the IGA mass. **Note that a carriage return is not added**. See also IGAMass.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for IGA mass igma:

```
var cards = igma.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last IGA Mass in the model.

Arguments

- **Model** (Model)

Model to get last IGA Mass in

Returns

IGAMass object (or null if there are no IGA Masss in the model).

Return type

IGAMass

Example

To get the last IGA Mass in model m:

```
var igma = IGAMass.Last(m);
```

Next()

Description

Returns the next IGA Mass in the model.

Arguments

No arguments

Returns

IGAMass object (or null if there are no more IGA Masss in the model).

Return type

IGAMass

Example

To get the IGA Mass in model m after IGA Mass igma:

```
var igma = igma.Next();
```

Previous()

Description

Returns the previous IGA Mass in the model.

Arguments

No arguments

Returns

IGAMass object (or null if there are no more IGA Masss in the model).

Return type

IGAMass

Example

To get the IGA Mass in model m before IGA Mass igma:

```
var igma = igma.Previous();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select IGA Masss using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting IGA Masss

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only IGA Masss from that model can be selected. If the argument is a Flag then only IGA Masss that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any IGA Masss can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of IGA Masss selected or null if menu cancelled

Return type

Number

Example

To select IGA Masss from model m, flagging those selected with flag f, giving the prompt 'Select IGA Masss':

```
IGAMass.Select(f, 'Select IGA Masss', m);
```

To select IGA Masss, flagging those selected with flag f but limiting selection to IGA Masss flagged with flag l, giving the prompt 'Select IGA Masss':

```
IGAMass.Select(f, 'Select IGA Masss', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the IGA Mass.

Arguments

- **flag** (Flag)

Flag to set on the IGA Mass

Returns

No return value

Example

To set flag f for IGA Mass igma:

```
igma.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the IGA Mass. The IGA Mass will be sketched until you either call IGAMass.Unsketch(), IGAMass.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Mass is sketched. If omitted redraw is true. If you want to sketch several IGA Masss and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch IGA Mass igma:

```
igma.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged IGA Masss in the model. The IGA Masss will be sketched until you either call IGAMass.Unsketch(), IGAMass.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged IGA Masss will be sketched in

- **flag** (Flag)

Flag set on the IGA Masss that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Masss are sketched. If omitted redraw is true. If you want to sketch flagged IGA Masss several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all IGA Masss flagged with flag in model m:

```
IGAMass.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of IGA Masss in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing IGA Masss should be counted. If false or omitted referenced but undefined IGA Masss will also be included in the total.

Returns

number of IGA Masss

Return type

Number

Example

To get the total number of IGA Masss in model m:

```
var total = IGAMass.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the IGA Masss in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all IGA Masss will be unset in

- **flag** (Flag)

Flag to unset on the IGA Masss

Returns

No return value

Example

To unset the flag f on all the IGA Masss in model m:

```
IGAMass.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the IGA Mass.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Mass is unsketched. If omitted redraw is true. If you want to unsketch several IGA Masss and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch IGA Mass igma:

```
igma.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all IGA Masss.

Arguments

- **Model** (Model)

Model that all IGA Masss will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Masss are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all IGA Masss in model m:

```
IGAMass.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged IGA Masss in the model.

Arguments

- **Model** (Model)

Model that all IGA Masss will be unsketched in

- **flag** (Flag)

Flag set on the IGA Masss that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Masss are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all IGA Masss flagged with flag in model m:

```
IGAMass.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

IGAMass object.

Return type

IGAMass

Example

To check if IGAMass property igma.example is a parameter by using the IGAMass.GetParameter() method:

```
if (igma.ViewParameters().GetParameter(igma.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for IGA Mass with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description

autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA Mass object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for IGA Mass igma:

```
igma.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for IGA Mass igma:

```
igma.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for IGA Mass. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for IGA Mass igma:

```
igma.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this IGA Mass.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for IGA Mass igma:

```
var xrefs = igma.Xrefs();
```

---

toString()

Description

Creates a string containing the IGA mass data in keyword format. Note that this contains the keyword header and the keyword cards. See also IGAMass.Keyword() and IGAMass.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for IGA mass igma in keyword format

```
var s = igma.toString();
```

IGAPointUVW class

The IGAPointUVW class gives you access to IGA point uvw cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]

- Xrefs()
- toString()

IGAPointUVW constants

Name	Description
IGAPointUVW.BASIS_TRANSFORM	BASIS_TRANSFORM is *IGA_POINT_UVW_BASIS_TRANSFORM.
IGAPointUVW.NONE	NONE is *IGA_POINT_UVW.

IGAPointUVW properties

Name	Type	Description
elid	integer	Element ID local to a basis transform.
exists (read only)	logical	true if IGA Point UVW exists, false if referred to but not defined.
include	integer	The Include file number that the IGA Point UVW is in.
label	integer	IGAPointUVW number. Also see the pid property which is an alternative name for this.
model (read only)	integer	The Model number that the IGA Point UVW is in.
nid	integer	Node ID.
option	constant	IGA Edge UVW option. • IGAPointUVW.NONE or • IGAPointUVW.BASIS_TRANSFORM
pid	integer	IGAPointUVW number. Also see the label property which is an alternative name for this.
u	real	Coordinates in the parametric u-direction.
v	real	Coordinates in the parametric v-direction.
w	real	Coordinates in the parametric w-direction.

Detailed Description

The IGAPointUVW class allows you to create, modify, edit and manipulate IGA point uvw cards. See the documentation below for more details.

Constructor

new IGAPointUVW(Model*[Model]*, details*[object]*)

Description

Create a new IGAPointUVW object.

Arguments

- **Model** (Model)

Model that IGA point uvw will be created in

- **details** (object)

Details for creating the IGAPointUVW

Object has the following properties:

Name	Type	Description
nid	integer	Node ID.

option (optional)	constant	IGA Point UVW option. IGAPointUVW.NONE or IGAPointUVW.BASIS_TRANSFORM
pid	integer	ID of the IGA Point UVW.
u (optional)	real	Coordinates in the parametric u-direction.
v (optional)	real	Coordinates in the parametric v-direction.
w (optional)	real	Coordinates in the parametric w-direction.

Returns

IGAPointUVW object

Return type

IGAPointUVW

Example

To create a new IGA point uvw in model m using pid 10 and node 100:

```
var igpu = new IGAPointUVW(m, {pid: 10, nid:100});
```

To create a new IGA point uvw in model m using pid 20 with node 200 and option BASIS_TRANSFORM and parametric coordinates u, v, w, as 3.3, 4.4, 5.5, elid 15 respectively:

```
var igpu = new IGAPointUVW(m, {option:IGAPointUVW.BASIS_TRANSFORM, pid: 20, nid: 200, u: 3.3, v: 4.4, w: 5.5, elid: 15});
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a IGA Point UVW.

Arguments

- **Comment** (Comment)

Comment that will be attached to the IGA Point UVW

Returns

No return value

Example

To associate comment c to the IGA Point UVW igpu:

```
igpu.AssociateComment(c);
```

Blank()

Description

Blanks the IGA Point UVW

Arguments

No arguments

Returns

No return value

Example

To blank IGA Point UVW igpu:

```
igpu.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the IGA Point UVWs in the model.

Arguments

- **Model** (Model)

Model that all IGA Point UVWs will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the IGA Point UVWs in model m:

```
IGAPointUVW.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged IGA Point UVWs in the model.

Arguments

- **Model** (Model)

Model that all the flagged IGA Point UVWs will be blanked in

- **flag** (Flag)

Flag set on the IGA Point UVWs that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the IGA Point UVWs in model m flagged with f:

```
IGAPointUVW.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the IGA Point UVW is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if IGA Point UVW igpu is blanked:

```
if (igpu.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse IGA Point UVW igpu:

```
igpu.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the IGA Point UVW.

Arguments

- **flag** (Flag)

Flag to clear on the IGA Point UVW

Returns

No return value

Example

To clear flag f for IGA Point UVW igpu:

```
igpu.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the IGA Point UVW. The target include of the copied IGA Point UVW can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

IGAPointUVW object

Return type

IGAPointUVW

Example

To copy IGA Point UVW igpu into IGA Point UVW z:

```
var z = igpu.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a IGA Point UVW

Arguments

- **Model** (Model)

Model that the IGA Point UVW will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

IGAPointUVW object (or null if not made).

Return type

IGAPointUVW

Example

To start creating a IGA Point UVW in model m:

```
var igpu = IGAPointUVW.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a IGA Point UVW.

Arguments

- **Comment** (Comment)

Comment that will be detached from the IGA Point UVW

Returns

No return value

Example

To detach comment c from the IGA Point UVW igpu:

```
igpu.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit IGA Point UVW igpu:

```
igpu.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for IGA Point UVW with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description

autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA Point UVW object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for IGA Point UVW igpu:

```
igpu.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for IGA Point UVW igpu:

```
igpu.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for IGA Point UVW. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for IGA Point UVW igpu:

```
igpu.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first IGA Point UVW in the model.

Arguments

- **Model** (Model)

Model to get first IGA Point UVW in

Returns

IGAPointUVW object (or null if there are no IGA Point UVWs in the model).

Return type

IGAPointUVW

Example

To get the first IGA Point UVW in model m:

```
var igpu = IGAPointUVW.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free IGA Point UVW label in the model. Also see IGAPointUVW.LastFreeLabel(), IGAPointUVW.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free IGA Point UVW label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

IGAPointUVW label.

Return type

Number

Example

To get the first free IGA Point UVW label in model m:

```
var label = IGAPointUVW.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the IGA Point UVWs in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all IGA Point UVWs will be flagged in

- **flag** (Flag)

Flag to set on the IGA Point UVWs

Returns

No return value

Example

To flag all of the IGA Point UVWs with flag f in model m:

```
IGAPointUVW.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the IGA Point UVW is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the IGA Point UVW

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if IGA Point UVW igpu has flag f set on it:

```
if (igpu.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each IGA Point UVW in the model.
Note that ForEach has been designed to make looping over IGA Point UVWs as fast as possible and so has some limitations.
Firstly, a single temporary IGAPointUVW object is created and on each function call it is updated with the current IGA Point UVW data. This means that you should not try to store the IGAPointUVW object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new IGA Point UVWs inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all IGA Point UVWs are in

- **func** (function)

Function to call for each IGA Point UVW

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the IGA Point UVWs in model m:

```
IGAPointUVW.ForEach(m, test);
function test(igpu)
{
// igpu is IGAPointUVW object
}
```

To call function test for all of the IGA Point UVWs in model m with optional object:

```
var data = { x:0, y:0 };
IGAPointUVW.ForEach(m, test, data);
function test(igpu, extra)
{
// igpu is IGAPointUVW object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of IGAPointUVW objects or properties for all of the IGA Point UVWs in a model in PRIMER. If the optional property argument is not given then an array of IGAPointUVW objects is returned. If the property argument is given, that property value for each IGA Point UVW is returned in the array instead of a IGAPointUVW object

Arguments

- **Model** (Model)

Model to get IGA Point UVWs from

- **property (optional)** (string)

Name for property to get for all IGA Point UVWs in the model

Returns

Array of IGAPointUVW objects or properties

Return type

Array

Example

To make an array of IGAPointUVW objects for all of the IGA Point UVWs in model m:

```
var a = IGAPointUVW.GetAll(m);
```

To return an array containing the value of property ’foo’ (for example ’x’ for a node) for each IGA Point UVW in model m:

```
var a = IGAPointUVW.GetAll(m, ’foo’);
```

GetComments()

Description

Extracts the comments associated to a IGA Point UVW.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the IGA Point UVW igpu:

```
var comm_array = igpu.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of IGAPointUVW objects for all of the flagged IGA Point UVWs in a model in PRIMER If the optional property argument is not given then an array of IGAPointUVW objects is returned. If the property argument is given, then that property value for each IGA Point UVW is returned in the array instead of a IGAPointUVW object

Arguments

- **Model** (Model)

Model to get IGA Point UVWs from

- **flag** (Flag)

Flag set on the IGA Point UVWs that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged IGA Point UVWs in the model

Returns

Array of IGAPointUVW objects or properties

Return type

Array

Example

To make an array of IGAPointUVW objects for all of the IGA Point UVWs in model m flagged with f:

```
var igpu = IGAPointUVW.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the IGA Point UVWs in model m flagged with f:

```
var a = IGAPointUVW.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the IGAPointUVW object for a IGA Point UVW ID.

Arguments

- **Model** (Model)

Model to find the IGA Point UVW in

- **number** (integer)

number of the IGA Point UVW you want the IGAPointUVW object for

Returns

IGAPointUVW object (or null if IGA Point UVW does not exist).

Return type

IGAPointUVW

Example

To get the IGAPointUVW object for IGA Point UVW 100 in model m

```
var igpu = IGAPointUVW.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a IGAPointUVW property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the IGAPointUVW.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

IGA Point UVW property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if IGAPointUVW property igpu.example is a parameter:

```
Options.property_parameter_names = true;
if (igpu.GetParameter(igpu.example) ) do_something...
Options.property_parameter_names = false;
```

To check if IGAPointUVW property igpu.example is a parameter by using the GetParameter method:

```
if (igpu.ViewParameters().GetParameter(igpu.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this IGA point uvw (*IGA_POINT_UVW). **Note that a carriage return is not added**. See also IGAPointUVW.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for IGA point uvw igpu:

```
var key = igpu.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the IGA point uvw. **Note that a carriage return is not added**. See also IGAPointUVW.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for IGA point uvw igpu:

```
var cards = igpu.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last IGA Point UVW in the model.

Arguments

- **Model** (Model)

Model to get last IGA Point UVW in

Returns

IGAPointUVW object (or null if there are no IGA Point UVWs in the model).

Return type

IGAPointUVW

Example

To get the last IGA Point UVW in model m:

```
var igpu = IGAPointUVW.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free IGA Point UVW label in the model. Also see IGAPointUVW.FirstFreeLabel(), IGAPointUVW.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free IGA Point UVW label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

IGAPointUVW label.

Return type

Number

Example

To get the last free IGA Point UVW label in model m:

```
var label = IGAPointUVW.LastFreeLabel(m);
```

Next()

Description

Returns the next IGA Point UVW in the model.

Arguments

No arguments

Returns

IGAPointUVW object (or null if there are no more IGA Point UVWs in the model).

Return type

IGAPointUVW

Example

To get the IGA Point UVW in model m after IGA Point UVW igpu:

```
var igpu = igpu.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) IGA Point UVW label in the model. Also see IGAPointUVW.FirstFreeLabel(), IGAPointUVW.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free IGA Point UVW label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

IGAPointUVW label.

Return type

Number

Example

To get the next free IGA Point UVW label in model m:

```
var label = IGAPointUVW.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a IGA Point UVW.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only IGA Point UVWs from that model can be picked. If the argument is a Flag then only IGA Point UVWs that are flagged with *limit* can be selected. If omitted, or null, any IGA Point UVWs from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

IGAPointUVW object (or null if not picked)

Return type

IGAPointUVW

Example

To pick a IGA Point UVW from model m giving the prompt 'Pick IGA Point UVW from screen':

```
var igpu = IGAPointUVW.Pick('Pick IGA Point UVW from screen', m);
```

Previous()

Description

Returns the previous IGA Point UVW in the model.

Arguments

No arguments

Returns

IGAPointUVW object (or null if there are no more IGA Point UVWs in the model).

Return type

IGAPointUVW

Example

To get the IGA Point UVW in model m before IGA Point UVW igpu:

```
var igpu = igpu.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the IGA Point UVWs in the model.

Arguments

- **Model** (Model)

Model that all IGA Point UVWs will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the IGA Point UVWs in model m, from 1000000:

```
IGAPointUVW.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged IGA Point UVWs in the model.

Arguments

- **Model** (Model)

Model that all the flagged IGA Point UVWs will be renumbered in

- **flag** (Flag)

Flag set on the IGA Point UVWs that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the IGA Point UVWs in model m flagged with f, from 1000000:

```
IGAPointUVW.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select IGA Point UVWs using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting IGA Point UVWs

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only IGA Point UVWs from that model can be selected. If the argument is a Flag then only IGA Point UVWs that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any IGA Point UVWs can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of IGA Point UVWs selected or null if menu cancelled

Return type

Number

Example

To select IGA Point UVWs from model m, flagging those selected with flag f, giving the prompt 'Select IGA Point UVWs':

```
IGAPointUVW.Select(f, 'Select IGA Point UVWs', m);
```

To select IGA Point UVWs, flagging those selected with flag f but limiting selection to IGA Point UVWs flagged with flag l, giving the prompt 'Select IGA Point UVWs':

```
IGAPointUVW.Select(f, 'Select IGA Point UVWs', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the IGA Point UVW.

Arguments

- **flag** (Flag)

Flag to set on the IGA Point UVW

Returns

No return value

Example

To set flag f for IGA Point UVW igpu:

```
igpu.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the IGA Point UVW. The IGA Point UVW will be sketched until you either call IGAPointUVW.Unsketch(), IGAPointUVW.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Point UVW is sketched. If omitted redraw is true. If you want to sketch several IGA Point UVWs and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch IGA Point UVW igpu:

```
igpu.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged IGA Point UVWs in the model. The IGA Point UVWs will be sketched until you either call IGAPointUVW.Unsketch(), IGAPointUVW.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged IGA Point UVWs will be sketched in

- **flag** (Flag)

Flag set on the IGA Point UVWs that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Point UVWs are sketched. If omitted redraw is true. If you want to sketch flagged IGA Point UVWs several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all IGA Point UVWs flagged with flag in model m:

```
IGAPointUVW.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of IGA Point UVWs in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing IGA Point UVWs should be counted. If false or omitted referenced but undefined IGA Point UVWs will also be included in the total.

Returns

number of IGA Point UVWs

Return type

Number

Example

To get the total number of IGA Point UVWs in model m:

```
var total = IGAPointUVW.Total(m);
```

Unblank()

Description

Unblanks the IGA Point UVW

Arguments

No arguments

Returns

No return value

Example

To unblank IGA Point UVW igpu:

```
igpu.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the IGA Point UVWs in the model.

Arguments

- **Model** (Model)

Model that all IGA Point UVWs will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the IGA Point UVWs in model m:

```
IGAPointUVW.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged IGA Point UVWs in the model.

Arguments

- **Model** (Model)

Model that the flagged IGA Point UVWs will be unblanked in

- **flag** (Flag)

Flag set on the IGA Point UVWs that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the IGA Point UVWs in model m flagged with f:

```
IGAPointUVW.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the IGA Point UVWs in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all IGA Point UVWs will be unset in

- **flag** (Flag)

Flag to unset on the IGA Point UVWs

Returns

No return value

Example

To unset the flag f on all the IGA Point UVWs in model m:

```
IGAPointUVW.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the IGA Point UVW.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Point UVW is unsketched. If omitted redraw is true. If you want to unsketch several IGA Point UVWs and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch IGA Point UVW igpu:

```
igpu.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all IGA Point UVWs.

Arguments

- **Model** (Model)

Model that all IGA Point UVWs will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Point UVWs are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all IGA Point UVWs in model m:

```
IGAPointUVW.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged IGA Point UVWs in the model.

Arguments

- **Model** (Model)

Model that all IGA Point UVWs will be unsketched in

- **flag** (Flag)

Flag set on the IGA Point UVWs that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Point UVWs are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all IGA Point UVWs flagged with flag in model m:

```
IGAPointUVW.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

IGAPointUVW object.

Return type

IGAPointUVW

Example

To check if IGAPointUVW property igpu.example is a parameter by using the IGAPointUVW.GetParameter() method:

```
if (igpu.ViewParameters().GetParameter(igpu.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for IGA Point UVW with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA Point UVW object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for IGA Point UVW igpu:

```
igpu.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for IGA Point UVW igpu:

```
igpu.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for IGA Point UVW. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for IGA Point UVW igpu:

```
igpu.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this IGA Point UVW.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for IGA Point UVW igpu:

```
var xrefs = igpu.Xrefs();
```

---

toString()

Description

Creates a string containing the IGA point uvw data in keyword format. Note that this contains the keyword header and the keyword cards. See also IGAPointUVW.Keyword() and IGAPointUVW.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for IGA point uvw igpu in keyword format

```
var s = igpu.toString();
```

IGARefineShell class

The IGARefineShell class gives you access to IGA refine shell cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

IGARefineShell properties

Name	Type	Description
exists (read only)	logical	true if IGA Refine Shell exists, false if referred to but not defined.
hrtyp	integer	IGA h-refinement type.
include	integer	The Include file number that the IGA Refine Shell is in.
label	integer	IGARefineShell ID. Also see the rid property which is an alternative name for this.
model (read only)	integer	The Model number that the IGA Refine Shell is in.

rid	integer	IGARefineShell ID. Also see the label property which is an alternative name for this.
rr	real	Number of equal parametric segments in the local r-direction.
rs	real	Number of equal parametric segments in the local s-direction.
rtyp	integer	Refinement type.
tr	integer	Target polynomial degree in the local r-direction.
ts	integer	Target polynomial degree in the local s-direction.

Detailed Description

The IGARefineShell class allows you to create, modify, edit and manipulate IGA refine shell cards. See the documentation below for more details.

Constructor

new IGARefineShell(Model*[Model]*, details*[object]*)

Description

Create a new IGARefineShell object.

Arguments

- **Model** (Model)

Model that IGA refine shell will be created in.

- **details** (object)

Details for creating the IGARefineShell

Object has the following properties:

Name	Type	Description
hrtyp (optional)	integer	IGA h-refinement type
rid	integer	IGA shell refinement ID
rr (optional)	real	Number of equal parametric segments in the local r-direction.
rs (optional)	real	Number of equal parametric segments in the local s-direction.
rtyp	integer	IGA refinement type
tr (optional)	integer	Target polynomial degree in the local r-direction.
ts (optional)	integer	Target polynomial degree in the local s-direction.

Returns

IGARefineShell object

Return type

IGARefineShell

Example

To create a new IGA refine shell in model m with rid 1, h-refinement 1 and rr and rs 2 and 4 respectively:

```
var igrsh = new IGARefineShell(m, {rid: 1, rtyp: 1, hrtyp: 1, rr: 2, rs: 4});
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a IGA Refine Shell.

Arguments

- **Comment** (Comment)

Comment that will be attached to the IGA Refine Shell

Returns

No return value

Example

To associate comment c to the IGA Refine Shell igrsh:

```
igrsh.AssociateComment(c);
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse IGA Refine Shell igrsh:

```
igrsh.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the IGA Refine Shell.

Arguments

- **flag** (Flag)

Flag to clear on the IGA Refine Shell

Returns

No return value

Example

To clear flag f for IGA Refine Shell igrsh:

```
igrsh.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the IGA Refine Shell. The target include of the copied IGA Refine Shell can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

IGARefineShell object

Return type

IGARefineShell

Example

To copy IGA Refine Shell igrsh into IGA Refine Shell z:

```
var z = igrsh.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a IGA Refine Shell

Arguments

- **Model** (Model)

Model that the IGA Refine Shell will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

IGARefineShell object (or null if not made).

Return type

IGARefineShell

Example

To start creating a IGA Refine Shell in model m:

```
var igrsh = IGARefineShell.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a IGA Refine Shell.

Arguments

- **Comment** (Comment)

Comment that will be detached from the IGA Refine Shell

Returns

No return value

Example

To detach comment c from the IGA Refine Shell igrsh:

```
igrsh.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit IGA Refine Shell igrsh:

```
igrsh.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for IGA Refine Shell with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description

autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA Refine Shell object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for IGA Refine Shell igrsh:

```
igrsh.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for IGA Refine Shell igrsh:

```
igrsh.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for IGA Refine Shell. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for IGA Refine Shell igrsh:

```
igrsh.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first IGA Refine Shell in the model.

Arguments

- **Model** (Model)

Model to get first IGA Refine Shell in

Returns

IGARefineShell object (or null if there are no IGA Refine Shells in the model).

Return type

IGARefineShell

Example

To get the first IGA Refine Shell in model m:

```
var igrsh = IGARefineShell.First(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the IGA Refine Shells in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all IGA Refine Shells will be flagged in

- **flag** (Flag)

Flag to set on the IGA Refine Shells

Returns

No return value

Example

To flag all of the IGA Refine Shells with flag f in model m:

```
IGARefineShell.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the IGA Refine Shell is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the IGA Refine Shell

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if IGA Refine Shell igrsh has flag f set on it:

```
if (igrsh.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each IGA Refine Shell in the model.
Note that ForEach has been designed to make looping over IGA Refine Shells as fast as possible and so has some limitations.
Firstly, a single temporary IGARefineShell object is created and on each function call it is updated with the current IGA Refine Shell data. This means that you should not try to store the IGARefineShell object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new IGA Refine Shells inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all IGA Refine Shells are in

- **func** (function)

Function to call for each IGA Refine Shell

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the IGA Refine Shells in model m:

```
IGARefineShell.ForEach(m, test);
function test(igrsh)
{
// igrsh is IGARefineShell object
}
```

To call function test for all of the IGA Refine Shells in model m with optional object:

```
var data = { x:0, y:0 };
IGARefineShell.ForEach(m, test, data);
function test(igrsh, extra)
{
// igrsh is IGARefineShell object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of IGARefineShell objects or properties for all of the IGA Refine Shells in a model in PRIMER. If the optional property argument is not given then an array of IGARefineShell objects is returned. If the property argument is given, that property value for each IGA Refine Shell is returned in the array instead of a IGARefineShell object

Arguments

- **Model** (Model)

Model to get IGA Refine Shells from

- **property (optional)** (string)

Name for property to get for all IGA Refine Shells in the model

Returns

Array of IGARefineShell objects or properties

Return type

Array

Example

To make an array of IGARefineShell objects for all of the IGA Refine Shells in model m:

```
var a = IGARefineShell.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each IGA Refine Shell in model m:

```
var a = IGARefineShell.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a IGA Refine Shell.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the IGA Refine Shell igrsh:

```
var comm_array = igrsh.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of IGARefineShell objects for all of the flagged IGA Refine Shells in a model in PRIMER If the optional property argument is not given then an array of IGARefineShell objects is returned. If the property argument is given, then that property value for each IGA Refine Shell is returned in the array instead of a IGARefineShell object

Arguments

- **Model** (Model)

Model to get IGA Refine Shells from

- **flag** (Flag)

Flag set on the IGA Refine Shells that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged IGA Refine Shells in the model

Returns

Array of IGARefineShell objects or properties

Return type

Array

Example

To make an array of IGARefineShell objects for all of the IGA Refine Shells in model m flagged with f:

```
var igrsh = IGARefineShell.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the IGA Refine Shells in model m flagged with f:

```
var a = IGARefineShell.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the IGARefineShell object for a IGA Refine Shell ID.

Arguments

- **Model** (Model)

Model to find the IGA Refine Shell in

- **number** (integer)

number of the IGA Refine Shell you want the IGARefineShell object for

Returns

IGARefineShell object (or null if IGA Refine Shell does not exist).

Return type

IGARefineShell

Example

To get the IGARefineShell object for IGA Refine Shell 100 in model m

```
var igrsh = IGARefineShell.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a IGARefineShell property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the IGARefineShell.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

IGA Refine Shell property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if IGARefineShell property igrsh.example is a parameter:

```
Options.property_parameter_names = true;
if (igrsh.GetParameter(igrsh.example) ) do_something...
Options.property_parameter_names = false;
```

To check if IGARefineShell property igrsh.example is a parameter by using the GetParameter method:

```
if (igrsh.ViewParameters().GetParameter(igrsh.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this IGA refine shell (*IGA_REFINE_SHELL). **Note that a carriage return is not added** See also IGARefineShell.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for IGA refine shell igs:

```
var key = igrsh.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the IGA refine shell. **Note that a carriage return is not added**. See also IGARefineShell.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for IGA refine shell igrsh:

```
var cards = igrsh.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last IGA Refine Shell in the model.

Arguments

- **Model** (Model)

Model to get last IGA Refine Shell in

Returns

IGARefineShell object (or null if there are no IGA Refine Shells in the model).

Return type

IGARefineShell

Example

To get the last IGA Refine Shell in model m:

```
var igrsh = IGARefineShell.Last(m);
```

Next()

Description

Returns the next IGA Refine Shell in the model.

Arguments

No arguments

Returns

IGARefineShell object (or null if there are no more IGA Refine Shells in the model).

Return type

IGARefineShell

Example

To get the IGA Refine Shell in model m after IGA Refine Shell igrsh:

```
var igrsh = igrsh.Next();
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a IGA Refine Shell.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only IGA Refine Shells from that model can be picked. If the argument is a Flag then only IGA Refine Shells that are flagged with *limit* can be selected. If omitted, or null, any IGA Refine Shells from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

IGARefineShell object (or null if not picked)

Return type

IGARefineShell

Example

To pick a IGA Refine Shell from model m giving the prompt 'Pick IGA Refine Shell from screen':

```
var igrsh = IGARefineShell.Pick('Pick IGA Refine Shell from screen', m);
```

Previous()

Description

Returns the previous IGA Refine Shell in the model.

Arguments

No arguments

Returns

IGARefineShell object (or null if there are no more IGA Refine Shells in the model).

Return type

IGARefineShell

Example

To get the IGA Refine Shell in model m before IGA Refine Shell igrsh:

```
var igrsh = igrsh.Previous();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select IGA Refine Shells using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting IGA Refine Shells

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only IGA Refine Shells from that model can be selected. If the argument is a Flag then only IGA Refine Shells that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any IGA Refine Shells can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of IGA Refine Shells selected or null if menu cancelled

Return type

Number

Example

To select IGA Refine Shells from model m, flagging those selected with flag f, giving the prompt 'Select IGA Refine Shells':

```
IGARefineShell.Select(f, 'Select IGA Refine Shells', m);
```

To select IGA Refine Shells, flagging those selected with flag f but limiting selection to IGA Refine Shells flagged with flag l, giving the prompt 'Select IGA Refine Shells':

```
IGARefineShell.Select(f, 'Select IGA Refine Shells', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the IGA Refine Shell.

Arguments

- **flag** (Flag)

Flag to set on the IGA Refine Shell

Returns

No return value

Example

To set flag f for IGA Refine Shell igrsh:

```
igrsh.SetFlag(f);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of IGA Refine Shells in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing IGA Refine Shells should be counted. If false or omitted referenced but undefined IGA Refine Shells will also be included in the total.

Returns

number of IGA Refine Shells

Return type

Number

Example

To get the total number of IGA Refine Shells in model m:

```
var total = IGARefineShell.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the IGA Refine Shells in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all IGA Refine Shells will be unset in

- **flag** (Flag)

Flag to unset on the IGA Refine Shells

Returns

No return value

Example

To unset the flag f on all the IGA Refine Shells in model m:

```
IGARefineShell.UnflagAll(m, f);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

IGARefineShell object.

Return type

IGARefineShell

Example

To check if IGARefineShell property igrsh.example is a parameter by using the IGARefineShell.GetParameter() method:

```
if (igrsh.ViewParameters().GetParameter(igrsh.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for IGA Refine Shell with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA Refine Shell object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for IGA Refine Shell igrsh:

```
igrsh.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for IGA Refine Shell igrsh:

```
igrsh.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for IGA Refine Shell. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for IGA Refine Shell igrsh:

```
igrsh.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this IGA Refine Shell.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for IGA Refine Shell igrsh:

```
var xrefs = igrsh.Xrefs();
```

---

toString()

Description

Creates a string containing the IGA refine shell data in keyword format. Note that this contains the keyword header and the keyword cards. See also IGARefineShell.Keyword() and IGARefineShell.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for IGA refine shell igrsh in keyword format

```
var s = igrsh.toString();
```

IGARefineSolid class

The IGARefineSolid class gives you access to IGA refine solid cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

IGARefineSolid properties

Name	Type	Description
exists (read only)	logical	true if IGA Refine Solid exists, false if referred to but not defined.
hrtyp	integer	IGA h-refinement type.
include	integer	The Include file number that the IGA Refine Solid is in.
label	integer	IGARefineSolid ID. Also see the rid property which is an alternative name for this.

model (read only)	integer	The Model number that the IGA Refine Solid is in.
rid	integer	IGARefineSolid ID. Also see the label property which is an alternative name for this.
rr	real	Number of equal parametric segments in the local r-direction or the physical length of knot spans in the local r-direction.
rs	real	Number of equal parametric segments in the local s-direction or the physical length of knot spans in the local s-direction.
rt	real	Number of equal parametric segments in the local t-direction or the physical length of knot spans in the local t-direction.
rtyp	integer	Refinement type.
tr	integer	Target polynomial degree in the local r-direction.
ts	integer	Target polynomial degree in the local s-direction.
tt	integer	Target polynomial degree in the local t-direction.

Detailed Description

The IGARefineSolid class allows you to create, modify, edit and manipulate IGA refine solid cards. See the documentation below for more details.

Constructor

new IGARefineSolid(Model*[Model]*, details*[object]*)

Description

Create a new IGARefineSolid object.

Arguments

- **Model** (Model)

Model that IGA refine solid will be created in.

- **details** (object)

Details for creating the IGARefineSolid

Object has the following properties:

Name	Type	Description
hrtyp (optional)	integer	IGA h-refinement type
rid	integer	IGA solid refinement ID
rr (optional)	real	Number of equal parametric segments in the local r-direction or the physical length of knot spans in the local r-direction.
rs (optional)	real	Number of equal parametric segments in the local s-direction or the physical length of knot spans in the local s-direction.
rt (optional)	real	Number of equal parametric segments in the local t-direction or the physical length of knot spans in the local t-direction.
rtyp	integer	IGA refinement type
tr (optional)	integer	Target polynomial degree in the local r-direction.
ts (optional)	integer	Target polynomial degree in the local s-direction.
tt (optional)	integer	Target polynomial degree in the local t-direction.

Returns

IGARefineSolid object

Return type

IGARefineSolid

Example

To create a new IGA refine solid in model m with rid 1, h-refinement 1, rr and rs rt 2, 4 and 6 respectively:

```
var igrso = new IGARefineSolid(m, {rid: 1, rtyp: 1, hrtyp: 1, rr: 2, rs: 4, rt: 6});
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a IGA Refine Solid.

Arguments

- **Comment** (Comment)

Comment that will be attached to the IGA Refine Solid

Returns

No return value

Example

To associate comment c to the IGA Refine Solid igrso:

```
igrso.AssociateComment(c);
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse IGA Refine Solid igrso:

```
igrso.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the IGA Refine Solid.

Arguments

- **flag** (Flag)

Flag to clear on the IGA Refine Solid

Returns

No return value

Example

To clear flag f for IGA Refine Solid igrso:

```
igrso.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the IGA Refine Solid. The target include of the copied IGA Refine Solid can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

IGARefineSolid object

Return type

IGARefineSolid

Example

To copy IGA Refine Solid igrso into IGA Refine Solid z:

```
var z = igrso.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a IGA Refine Solid

Arguments

- **Model** (Model)

Model that the IGA Refine Solid will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

IGARefineSolid object (or null if not made).

Return type

IGARefineSolid

Example

To start creating a IGA Refine Solid in model m:

```
var igrso = IGARefineSolid.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a IGA Refine Solid.

Arguments

- **Comment** (Comment)

Comment that will be detached from the IGA Refine Solid

Returns

No return value

Example

To detach comment c from the IGA Refine Solid igrso:

```
igrso.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit IGA Refine Solid igrso:

```
igrso.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for IGA Refine Solid with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA Refine Solid object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for IGA Refine Solid igrso:

```
igrso.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for IGA Refine Solid igrso:

```
igrso.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for IGA Refine Solid. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for IGA Refine Solid igrso:

```
igrso.Error("My custom error");
```

First(Model[*Model*]) [static]

Description

Returns the first IGA Refine Solid in the model.

Arguments

- **Model** (Model)

Model to get first IGA Refine Solid in

Returns

IGARefineSolid object (or null if there are no IGA Refine Solids in the model).

Return type

IGARefineSolid

Example

To get the first IGA Refine Solid in model m:

```
var igrso = IGARefineSolid.First(m);
```

FlagAll(Model[*Model*], flag[*Flag*]) [static]

Description

Flags all of the IGA Refine Solids in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all IGA Refine Solids will be flagged in

- **flag** (Flag)

Flag to set on the IGA Refine Solids

Returns

No return value

Example

To flag all of the IGA Refine Solids with flag f in model m:

```
IGARefineSolid.FlagAll(m, f);
```

Flagged(flag[*Flag*])

Description

Checks if the IGA Refine Solid is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the IGA Refine Solid

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if IGA Refine Solid igrso has flag f set on it:

```
if (igrso.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each IGA Refine Solid in the model.
Note that ForEach has been designed to make looping over IGA Refine Solids as fast as possible and so has some limitations.
Firstly, a single temporary IGARefineSolid object is created and on each function call it is updated with the current IGA Refine Solid data. This means that you should not try to store the IGARefineSolid object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new IGA Refine Solids inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all IGA Refine Solids are in

- **func** (function)

Function to call for each IGA Refine Solid

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the IGA Refine Solids in model m:

```
IGARefineSolid.ForEach(m, test);
function test(igrso)
{
// igrso is IGARefineSolid object
}
```

To call function test for all of the IGA Refine Solids in model m with optional object:

```
var data = { x:0, y:0 };
IGARefineSolid.ForEach(m, test, data);
function test(igrso, extra)
{
// igrso is IGARefineSolid object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of IGARefineSolid objects or properties for all of the IGA Refine Solids in a model in PRIMER. If the optional property argument is not given then an array of IGARefineSolid objects is returned. If the property argument is given, that property value for each IGA Refine Solid is returned in the array instead of a IGARefineSolid object

Arguments

- **Model** (Model)

Model to get IGA Refine Solids from

- **property (optional)** (string)

Name for property to get for all IGA Refine Solids in the model

Returns

Array of IGARefineSolid objects or properties

Return type

Array

Example

To make an array of IGARefineSolid objects for all of the IGA Refine Solids in model m:

```
var a = IGARefineSolid.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each IGA Refine Solid in model m:

```
var a = IGARefineSolid.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a IGA Refine Solid.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the IGA Refine Solid igrso:

```
var comm_array = igrso.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of IGARefineSolid objects for all of the flagged IGA Refine Solids in a model in PRIMER If the optional property argument is not given then an array of IGARefineSolid objects is returned. If the property argument is given, then that property value for each IGA Refine Solid is returned in the array instead of a IGARefineSolid object

Arguments

- **Model** (Model)

Model to get IGA Refine Solids from

- **flag** (Flag)

Flag set on the IGA Refine Solids that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged IGA Refine Solids in the model

Returns

Array of IGARefineSolid objects or properties

Return type

Array

Example

To make an array of IGARefineSolid objects for all of the IGA Refine Solids in model m flagged with f:

```
var igrso = IGARefineSolid.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the IGA Refine Solids in model m flagged with f:

```
var a = IGARefineSolid.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the IGARefineSolid object for a IGA Refine Solid ID.

Arguments

- **Model** (Model)

Model to find the IGA Refine Solid in

- **number** (integer)

number of the IGA Refine Solid you want the IGARefineSolid object for

Returns

IGARefineSolid object (or null if IGA Refine Solid does not exist).

Return type

IGARefineSolid

Example

To get the IGARefineSolid object for IGA Refine Solid 100 in model m

```
var igrso = IGARefineSolid.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a IGARefineSolid property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the IGARefineSolid.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

IGA Refine Solid property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if IGARefineSolid property igrso.example is a parameter:

```
Options.property_parameter_names = true;
if (igrso.GetParameter(igrso.example) ) do_something...
Options.property_parameter_names = false;
```

To check if IGARefineSolid property igrso.example is a parameter by using the GetParameter method:

```
if (igrso.ViewParameters().GetParameter(igrso.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this IGA refine solid (*IGA_REFINE_SOLID). **Note that a carriage return is not added** See also IGARefineSolid.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for IGA refine solid igrso:

```
var key = igrso.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the IGA refine solid. **Note that a carriage return is not added**. See also IGARefineSolid.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for IGA refine solid igrso:

```
var cards = igrso.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last IGA Refine Solid in the model.

Arguments

- **Model** (Model)

Model to get last IGA Refine Solid in

Returns

IGARefineSolid object (or null if there are no IGA Refine Solids in the model).

Return type

IGARefineSolid

Example

To get the last IGA Refine Solid in model m:

```
var igrso = IGARefineSolid.Last(m);
```

Next()

Description

Returns the next IGA Refine Solid in the model.

Arguments

No arguments

Returns

IGARefineSolid object (or null if there are no more IGA Refine Solids in the model).

Return type

IGARefineSolid

Example

To get the IGA Refine Solid in model m after IGA Refine Solid igrso:

```
var igrso = igrso.Next();
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a IGA Refine Solid.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only IGA Refine Solids from that model can be picked. If the argument is a Flag then only IGA Refine Solids that are flagged with *limit* can be selected. If omitted, or null, any IGA Refine Solids from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

IGARefineSolid object (or null if not picked)

Return type

IGARefineSolid

Example

To pick a IGA Refine Solid from model m giving the prompt 'Pick IGA Refine Solid from screen':

```
var igrso = IGARefineSolid.Pick('Pick IGA Refine Solid from screen', m);
```

Previous()

Description

Returns the previous IGA Refine Solid in the model.

Arguments

No arguments

Returns

IGARefineSolid object (or null if there are no more IGA Refine Solids in the model).

Return type

IGARefineSolid

Example

To get the IGA Refine Solid in model m before IGA Refine Solid igrso:

```
var igrso = igrso.Previous();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select IGA Refine Solids using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting IGA Refine Solids

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only IGA Refine Solids from that model can be selected. If the argument is a Flag then only IGA Refine Solids that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any IGA Refine Solids can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of IGA Refine Solids selected or null if menu cancelled

Return type

Number

Example

To select IGA Refine Solids from model m, flagging those selected with flag f, giving the prompt 'Select IGA Refine Solids':

```
IGARefineSolid.Select(f, 'Select IGA Refine Solids', m);
```

To select IGA Refine Solids, flagging those selected with flag f but limiting selection to IGA Refine Solids flagged with flag l, giving the prompt 'Select IGA Refine Solids':

```
IGARefineSolid.Select(f, 'Select IGA Refine Solids', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the IGA Refine Solid.

Arguments

- **flag** (Flag)

Flag to set on the IGA Refine Solid

Returns

No return value

Example

To set flag f for IGA Refine Solid igrso:

```
igrso.SetFlag(f);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of IGA Refine Solids in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing IGA Refine Solids should be counted. If false or omitted referenced but undefined IGA Refine Solids will also be included in the total.

Returns

number of IGA Refine Solids

Return type

Number

Example

To get the total number of IGA Refine Solids in model m:

```
var total = IGARefineSolid.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the IGA Refine Solids in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all IGA Refine Solids will be unset in

- **flag** (Flag)

Flag to unset on the IGA Refine Solids

Returns

No return value

Example

To unset the flag f on all the IGA Refine Solids in model m:

```
IGARefineSolid.UnflagAll(m, f);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

IGARefineSolid object.

Return type

IGARefineSolid

Example

To check if IGARefineSolid property igrso.example is a parameter by using the IGARefineSolid.GetParameter() method:

```
if (igrso.ViewParameters().GetParameter(igrso.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for IGA Refine Solid with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA Refine Solid object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for IGA Refine Solid igrso:

```
igrso.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for IGA Refine Solid igrso:

```
igrso.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for IGA Refine Solid. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for IGA Refine Solid igrso:

```
igrso.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this IGA Refine Solid.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for IGA Refine Solid igrso:

```
var xrefs = igrso.Xrefs();
```

---

toString()

Description

Creates a string containing the IGA refine solid data in keyword format. Note that this contains the keyword header and the keyword cards. See also IGARefineSolid.Keyword() and IGARefineSolid.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for IGA refine solid igrso in keyword format

```
var s = igrso.toString();
```

IGAShell class

The IGAShell class gives you access to IGA shell cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model *[Model]*, redraw (optional) *[boolean]*)
- BlankFlagged(Model *[Model]*, flag *[Flag]*, redraw (optional) *[boolean]*)
- Create(Model *[Model]*, modal (optional) *[boolean]*)
- First(Model *[Model]*)
- FlagAll(Model *[Model]*, flag *[Flag]*)
- ForEach(Model *[Model]*, func *[function]*, extra (optional) *[any]*)
- GetAll(Model *[Model]*, property (optional) *[string]*)
- GetFlagged(Model *[Model]*, flag *[Flag]*, property (optional) *[string]*)
- GetFromID(Model *[Model]*, number *[integer]*)
- Last(Model *[Model]*)
- Pick(prompt *[string]*, limit (optional) *[Model or Flag]*, modal (optional) *[boolean]*, button text (optional) *[string]*)
- RedrawAll(Model *[Model]*)
- Select(flag *[Flag]*, prompt *[string]*, limit (optional) *[Model or Flag]*, modal (optional) *[boolean]*)
- SketchFlagged(Model *[Model]*, flag *[Flag]*, redraw (optional) *[boolean]*)
- Total(Model *[Model]*, exists (optional) *[boolean]*)
- UnblankAll(Model *[Model]*, redraw (optional) *[boolean]*)
- UnblankFlagged(Model *[Model]*, flag *[Flag]*, redraw (optional) *[boolean]*)
- UnflagAll(Model *[Model]*, flag *[Flag]*)
- UnsketchAll(Model *[Model]*, redraw (optional) *[boolean]*)
- UnsketchFlagged(Model *[Model]*, flag *[Flag]*, redraw (optional) *[boolean]*)

Member functions

- AssociateComment(Comment *[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional) *[boolean]*)
- ClearFlag(flag *[Flag]*)
- Copy(range (optional) *[boolean]*)
- DetachComment(Comment *[Comment]*)
- Edit(modal (optional) *[boolean]*)
- Error(message *[string]*, options (optional) *[object]*)
- Error(message *[string]*, details (optional) *[string]*) [deprecated]
- Flagged(flag *[Flag]*)
- GetComments()
- GetParameter(prop *[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag *[Flag]*)
- Sketch(redraw (optional) *[boolean]*)
- Unblank()
- Unsketch(redraw (optional) *[boolean]*)
- ViewParameters()
- Warning(message *[string]*, options (optional) *[object]*)
- Warning(message *[string]*, details (optional) *[string]*) [deprecated]
- Xrefs()
- toString()

IGAShell properties

Name	Type	Description
exists (read only)	logical	true if IGA Shell exists, false if referred to but not defined.
id (read only)	integer	ID of the IGA shell. Only used in PRIMER.
idfne	integer	Element ID of the first IGA element.
include	integer	The Include file number that the IGA Shell is in.
model (read only)	integer	The Model number that the IGA Shell is in.
nisr	real	Interpolation elements in the local r-direction.
niss	real	Interpolation elements in the local s-direction.
pid	integer	Part ID.
rid	integer	IGA refine shell ID.
sid	integer	Isogeometric Shell (patch) ID.

Detailed Description

The IGAShell class allows you to create, modify, edit and manipulate IGA shell cards. See the documentation below for more details.

Constructor

new IGAShell(Model*[Model]*, details*[object]*)

Description

Create a new IGAShell object.

Arguments

- **Model** (Model)

Model that IGA shell will be created in

- **details** (object)

Details for creating the IGAShell

Object has the following properties:

Name	Type	Description
idfne (optional)	integer	Element ID of the first IGA element.
nisr (optional)	real	Interpolation elements in the local r-direction.
niss (optional)	real	Interpolation elements in the local s-direction.
pid	integer	Part ID.
rid (optional)	integer	IGA refine shell ID.
sid	integer	Isogeometric Shell (patch) ID.

Returns

IGAShell object

Return type

IGAShell

Example

To create a new IGA shell in model m using sid 10, part id 100:

```
var igsh = new IGAShell(m, {sid: 10, pid: 100});
```

To create a new IGA shell in model m using sid 20, part id 200 with nisr as 3.3, niss as 4.4 and idfne as 50:

```
var igsh = new IGAShell(m, {sid: 20, pid: 200, nisr: 3.3, niss: 4.4, idfne:
50});
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a IGA Shell.

Arguments

- **Comment** (Comment)

Comment that will be attached to the IGA Shell

Returns

No return value

Example

To associate comment c to the IGA Shell igsh:

```
igsh.AssociateComment(c);
```

Blank()

Description

Blanks the IGA Shell

Arguments

No arguments

Returns

No return value

Example

To blank IGA Shell igsh:

```
igsh.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the IGA Shells in the model.

Arguments

- **Model** (Model)

Model that all IGA Shells will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the IGA Shells in model m:

```
IGAShell.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged IGA Shells in the model.

Arguments

- **Model** (Model)

Model that all the flagged IGA Shells will be blanked in

- **flag** (Flag)

Flag set on the IGA Shells that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the IGA Shells in model m flagged with f:

```
IGAShell.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the IGA Shell is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if IGA Shell igsh is blanked:

```
if (igsh.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse IGA Shell igsh:

```
igsh.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the IGA Shell.

Arguments

- **flag** (Flag)

Flag to clear on the IGA Shell

Returns

No return value

Example

To clear flag f for IGA Shell igsh:

```
igsh.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the IGA Shell. The target include of the copied IGA Shell can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

IGAShell object

Return type

IGAShell

Example

To copy IGA Shell igsh into IGA Shell z:

```
var z = igsh.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a IGA Shell

Arguments

- **Model** (Model)

Model that the IGA Shell will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

IGAShell object (or null if not made).

Return type

IGAShell

Example

To start creating a IGA Shell in model m:

```
var igsh = IGAShell.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a IGA Shell.

Arguments

- **Comment** (Comment)

Comment that will be detached from the IGA Shell

Returns

No return value

Example

To detach comment c from the IGA Shell igsh:

```
igsh.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit IGA Shell igsh:

```
igsh.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for IGA Shell with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA Shell object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for IGA Shell igsh:

```
igsh.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for IGA Shell igsh:

```
igsh.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for IGA Shell. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for IGA Shell igsh:

```
igsh.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first IGA Shell in the model.

Arguments

- **Model** (Model)

Model to get first IGA Shell in

Returns

IGAShell object (or null if there are no IGA Shells in the model).

Return type

IGAShell

Example

To get the first IGA Shell in model m:

```
var igsh = IGAShell.First(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the IGA Shells in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all IGA Shells will be flagged in

- **flag** (Flag)

Flag to set on the IGA Shells

Returns

No return value

Example

To flag all of the IGA Shells with flag f in model m:

```
IGAShell.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the IGA Shell is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the IGA Shell

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if IGA Shell igsh has flag f set on it:

```
if (igsh.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each IGA Shell in the model.
Note that ForEach has been designed to make looping over IGA Shells as fast as possible and so has some limitations.
Firstly, a single temporary IGAShell object is created and on each function call it is updated with the current IGA Shell data. This means that you should not try to store the IGAShell object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new IGA Shells inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all IGA Shells are in

- **func** (function)

Function to call for each IGA Shell

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the IGA Shells in model m:

```
IGAShell.ForEach(m, test);
function test(igsh)
{
// igsh is IGAShell object
}
```

To call function test for all of the IGA Shells in model m with optional object:

```
var data = { x:0, y:0 };
IGAShell.ForEach(m, test, data);
function test(igsh, extra)
{
// igsh is IGAShell object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of IGAShell objects or properties for all of the IGA Shells in a model in PRIMER. If the optional property argument is not given then an array of IGAShell objects is returned. If the property argument is given, that property value for each IGA Shell is returned in the array instead of a IGAShell object

Arguments

- **Model** (Model)

Model to get IGA Shells from

- **property (optional)** (string)

Name for property to get for all IGA Shells in the model

Returns

Array of IGAShell objects or properties

Return type

Array

Example

To make an array of IGAShell objects for all of the IGA Shells in model m:

```
var a = IGAShell.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each IGA Shell in model m:

```
var a = IGAShell.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a IGA Shell.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the IGA Shell igsh:

```
var comm_array = igsh.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of IGAShell objects for all of the flagged IGA Shells in a model in PRIMER If the optional property argument is not given then an array of IGAShell objects is returned. If the property argument is given, then that property value for each IGA Shell is returned in the array instead of a IGAShell object

Arguments

- **Model** (Model)

Model to get IGA Shells from

- **flag** (Flag)

Flag set on the IGA Shells that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged IGA Shells in the model

Returns

Array of IGAShell objects or properties

Return type

Array

Example

To make an array of IGAShell objects for all of the IGA Shells in model m flagged with f:

```
var igsh = IGAShell.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the IGA Shells in model m flagged with f:

```
var a = IGAShell.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the IGAShell object for a IGA Shell ID.

Arguments

- **Model** (Model)

Model to find the IGA Shell in

- **number** (integer)

number of the IGA Shell you want the IGAShell object for

Returns

IGAShell object (or null if IGA Shell does not exist).

Return type

IGAShell

Example

To get the IGAShell object for IGA Shell 100 in model m

```
var igsh = IGAShell.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a IGAShell property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the IGAShell.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

IGA Shell property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if IGAShell property igsh.example is a parameter:

```
Options.property_parameter_names = true;
if (igsh.GetParameter(igsh.example) ) do_something...
Options.property_parameter_names = false;
```

To check if IGAShell property igsh.example is a parameter by using the GetParameter method:

```
if (igsh.ViewParameters().GetParameter(igsh.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this IGA shell (*IGA_SHELL). **Note that a carriage return is not added**. See also IGAShell.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for IGA shell igs:

```
var key = igs.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the IGA shell. **Note that a carriage return is not added**. See also IGAShell.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for IGA shell igs:

```
var cards = igs.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last IGA Shell in the model.

Arguments

- **Model** (Model)

Model to get last IGA Shell in

Returns

IGAShell object (or null if there are no IGA Shells in the model).

Return type

IGAShell

Example

To get the last IGA Shell in model m:

```
var igsh = IGAShell.Last(m);
```

Next()

Description

Returns the next IGA Shell in the model.

Arguments

No arguments

Returns

IGAShell object (or null if there are no more IGA Shells in the model).

Return type

IGAShell

Example

To get the IGA Shell in model m after IGA Shell igsh:

```
var igsh = igsh.Next();
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a IGA Shell.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only IGA Shells from that model can be picked. If the argument is a Flag then only IGA Shells that are flagged with *limit* can be selected. If omitted, or null, any IGA Shells from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

IGAShell object (or null if not picked)

Return type

IGAShell

Example

To pick a IGA Shell from model m giving the prompt 'Pick IGA Shell from screen':

```
var igsh = IGAShell.Pick('Pick IGA Shell from screen', m);
```

Previous()

Description

Returns the previous IGA Shell in the model.

Arguments

No arguments

Returns

IGAShell object (or null if there are no more IGA Shells in the model).

Return type

IGAShell

Example

To get the IGA Shell in model m before IGA Shell igsh:

```
var igsh = igsh.Previous();
```

RedrawAll(Model*[Model]*) [static]

Description

Redraws all *IGA_SHELLs whose graphics are out of date in a specified model

Arguments

- **Model** (Model)

Model to redraw IGA SHELLs in

Returns

No return value

Example

Redraw all out of date *IGA_SHELLs following a change to an *IGA_SHELL or related IGA entity in model m1:

```
IGAShell.RedrawAll(m1);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select IGA Shells using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting IGA Shells

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only IGA Shells from that model can be selected. If the argument is a Flag then only IGA Shells that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any IGA Shells can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of IGA Shells selected or null if menu cancelled

Return type

Number

Example

To select IGA Shells from model m, flagging those selected with flag f, giving the prompt 'Select IGA Shells':

```
IGAShell.Select(f, 'Select IGA Shells', m);
```

To select IGA Shells, flagging those selected with flag f but limiting selection to IGA Shells flagged with flag l, giving the prompt 'Select IGA Shells':

```
IGAShell.Select(f, 'Select IGA Shells', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the IGA Shell.

Arguments

- **flag** (Flag)

Flag to set on the IGA Shell

Returns

No return value

Example

To set flag f for IGA Shell igsh:

```
igsh.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the IGA Shell. The IGA Shell will be sketched until you either call IGAShell.Unsketch(), IGAShell.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Shell is sketched. If omitted redraw is true. If you want to sketch several IGA Shells and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch IGA Shell igsh:

```
igsh.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged IGA Shells in the model. The IGA Shells will be sketched until you either call IGAShell.Unsketch(), IGAShell.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged IGA Shells will be sketched in

- **flag** (Flag)

Flag set on the IGA Shells that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Shells are sketched. If omitted redraw is true. If you want to sketch flagged IGA Shells several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all IGA Shells flagged with flag in model m:

```
IGAShell.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of IGA Shells in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing IGA Shells should be counted. If false or omitted referenced but undefined IGA Shells will also be included in the total.

Returns

number of IGA Shells

Return type

Number

Example

To get the total number of IGA Shells in model m:

```
var total = IGAShell.Total(m);
```

Unblank()

Description

Unblanks the IGA Shell

Arguments

No arguments

Returns

No return value

Example

To unblank IGA Shell igsh:

```
igsh.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the IGA Shells in the model.

Arguments

- **Model** (Model)

Model that all IGA Shells will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the IGA Shells in model m:

```
IGAShell.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged IGA Shells in the model.

Arguments

- **Model** (Model)

Model that the flagged IGA Shells will be unblanked in

- **flag** (Flag)

Flag set on the IGA Shells that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the IGA Shells in model m flagged with f:

```
IGAShell.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the IGA Shells in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all IGA Shells will be unset in

- **flag** (Flag)

Flag to unset on the IGA Shells

Returns

No return value

Example

To unset the flag f on all the IGA Shells in model m:

```
IGAShell.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the IGA Shell.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Shell is unsketched. If omitted redraw is true. If you want to unsketch several IGA Shells and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch IGA Shell igsh:

```
igsh.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all IGA Shells.

Arguments

- **Model** (Model)

Model that all IGA Shells will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Shells are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all IGA Shells in model m:

```
IGAShell.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged IGA Shells in the model.

Arguments

- **Model** (Model)

Model that all IGA Shells will be unsketched in

- **flag** (Flag)

Flag set on the IGA Shells that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Shells are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all IGA Shells flagged with flag in model m:

```
IGAShell.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with ’method chaining’ (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

IGAShell object.

Return type

IGAShell

Example

To check if IGAShell property igsh.example is a parameter by using the IGAShell.GetParameter() method:

```
if (igsh.ViewParameters().GetParameter(igsh.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for IGA Shell with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA Shell object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for IGA Shell igsh:

```
igsh.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for IGA Shell igsh:

```
igsh.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for IGA Shell. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for IGA Shell igsh:

```
igsh.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this IGA Shell.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for IGA Shell igsh:

```
var xrefs = igsh.Xrefs();
```

---

toString()

Description

Creates a string containing the IGA shell data in keyword format. Note that this contains the keyword header and the keyword cards. See also IGAShell.Keyword() and IGAShell.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for IGA shell igs in keyword format

```
var s = igs.toString();
```

IGASolid class

The IGASolid class gives you access to IGA solid cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model *[Model]*, redraw (optional) *[boolean]*)
- BlankFlagged(Model *[Model]*, flag *[Flag]*, redraw (optional) *[boolean]*)
- Create(Model *[Model]*, modal (optional) *[boolean]*)
- First(Model *[Model]*)
- FlagAll(Model *[Model]*, flag *[Flag]*)
- ForEach(Model *[Model]*, func *[function]*, extra (optional) *[any]*)
- GetAll(Model *[Model]*, property (optional) *[string]*)
- GetFlagged(Model *[Model]*, flag *[Flag]*, property (optional) *[string]*)
- GetFromID(Model *[Model]*, number *[integer]*)
- Last(Model *[Model]*)
- Pick(prompt *[string]*, limit (optional) *[Model or Flag]*, modal (optional) *[boolean]*, button text (optional) *[string]*)
- RedrawAll(Model *[Model]*)
- Select(flag *[Flag]*, prompt *[string]*, limit (optional) *[Model or Flag]*, modal (optional) *[boolean]*)
- SketchFlagged(Model *[Model]*, flag *[Flag]*, redraw (optional) *[boolean]*)
- Total(Model *[Model]*, exists (optional) *[boolean]*)
- UnblankAll(Model *[Model]*, redraw (optional) *[boolean]*)
- UnblankFlagged(Model *[Model]*, flag *[Flag]*, redraw (optional) *[boolean]*)
- UnflagAll(Model *[Model]*, flag *[Flag]*)
- UnsketchAll(Model *[Model]*, redraw (optional) *[boolean]*)
- UnsketchFlagged(Model *[Model]*, flag *[Flag]*, redraw (optional) *[boolean]*)

Member functions

- AssociateComment(Comment *[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional) *[boolean]*)
- ClearFlag(flag *[Flag]*)
- Copy(range (optional) *[boolean]*)
- DetachComment(Comment *[Comment]*)
- Edit(modal (optional) *[boolean]*)
- Error(message *[string]*, options (optional) *[object]*)
- Error(message *[string]*, details (optional) *[string]*) [deprecated]
- Flagged(flag *[Flag]*)
- GetComments()
- GetParameter(prop *[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag *[Flag]*)
- Sketch(redraw (optional) *[boolean]*)
- Unblank()
- Unsketch(redraw (optional) *[boolean]*)
- ViewParameters()
- Warning(message *[string]*, options (optional) *[object]*)
- Warning(message *[string]*, details (optional) *[string]*) [deprecated]
- Xrefs()
- toString()

IGASolid properties

Name	Type	Description
exists (read only)	logical	true if IGA Solid exists, false if referred to but not defined.
id (read only)	integer	ID of the IGA solid. Only used in PRIMER.
include	integer	The Include file number that the IGA Solid is in.
model (read only)	integer	The Model number that the IGA Solid is in.
nisr	real	Interpolation elements in the local r-direction.
niss	real	Interpolation elements in the local s-direction.
nist	real	Interpolation elements in the local t-direction.
pid	integer	Part ID.
rid	integer	Solid refinement ID.
sid	integer	Isogeometric Solid (patch) ID.

Detailed Description

The IGASolid class allows you to create, modify, edit and manipulate IGA solid cards. See the documentation below for more details.

Constructor

new IGASolid(Model*[Model]*, details*[object]*)

Description

Create a new IGASolid object.

Arguments

- **Model** (Model)

Model that IGA solid will be created in

- **details** (object)

Details for creating the IGASolid

Object has the following properties:

Name	Type	Description
nisr (optional)	real	Interpolation elements in the local r-direction.
niss (optional)	real	Interpolation elements in the local s-direction.
nist (optional)	real	Interpolation elements in the local t-direction.
pid	integer	Part ID.
rid (optional)	integer	Solid refinement ID.
sid	integer	Isogeometric Solid (patch) ID.

Returns

IGASolid object

Return type

IGASolid

Example

To create a new IGA solid in model m using sid 10, part id 100:

```
var igso = new IGASolid(m, {sid: 10, pid: 100});
```

To create a new IGA solid in model m using sid 20, part id 200 with nisr as 3.3, niss as 4.4, nist as 5.5 and rid as 6:

```
var igso = new IGASolid(m, {sid: 20, pid: 200, nisr: 3.3, niss: 4.4, nist: 5.5,
rid: 6});
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a IGA Solid.

Arguments

- **Comment** (Comment)

Comment that will be attached to the IGA Solid

Returns

No return value

Example

To associate comment c to the IGA Solid igso:

```
igso.AssociateComment(c);
```

Blank()

Description

Blanks the IGA Solid

Arguments

No arguments

Returns

No return value

Example

To blank IGA Solid igso:

```
igso.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the IGA Solids in the model.

Arguments

- **Model** (Model)

Model that all IGA Solids will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the IGA Solids in model m:

```
IGASolid.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged IGA Solids in the model.

Arguments

- **Model** (Model)

Model that all the flagged IGA Solids will be blanked in

- **flag** (Flag)

Flag set on the IGA Solids that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the IGA Solids in model m flagged with f:

```
IGASolid.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the IGA Solid is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if IGA Solid igso is blanked:

```
if (igso.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse IGA Solid igso:

```
igso.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the IGA Solid.

Arguments

- **flag** (Flag)

Flag to clear on the IGA Solid

Returns

No return value

Example

To clear flag f for IGA Solid igso:

```
igso.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the IGA Solid. The target include of the copied IGA Solid can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

IGASolid object

Return type

IGASolid

Example

To copy IGA Solid igso into IGA Solid z:

```
var z = igso.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a IGA Solid

Arguments

- **Model** (Model)

Model that the IGA Solid will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

IGASolid object (or null if not made).

Return type

IGASolid

Example

To start creating a IGA Solid in model m:

```
var igso = IGASolid.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a IGA Solid.

Arguments

- **Comment** (Comment)

Comment that will be detached from the IGA Solid

Returns

No return value

Example

To detach comment c from the IGA Solid igso:

```
igso.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit IGA Solid igso:

```
igso.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for IGA Solid with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA Solid object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for IGA Solid igso:

```
igso.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for IGA Solid igso:

```
igso.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for IGA Solid. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for IGA Solid igso:

```
igso.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first IGA Solid in the model.

Arguments

- **Model** (Model)

Model to get first IGA Solid in

Returns

IGASolid object (or null if there are no IGA Solids in the model).

Return type

IGASolid

Example

To get the first IGA Solid in model m:

```
var igso = IGASolid.First(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the IGA Solids in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all IGA Solids will be flagged in

- **flag** (Flag)

Flag to set on the IGA Solids

Returns

No return value

Example

To flag all of the IGA Solids with flag f in model m:

```
IGASolid.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the IGA Solid is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the IGA Solid

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if IGA Solid igso has flag f set on it:

```
if (igso.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each IGA Solid in the model.
Note that ForEach has been designed to make looping over IGA Solids as fast as possible and so has some limitations.
Firstly, a single temporary IGASolid object is created and on each function call it is updated with the current IGA Solid data. This means that you should not try to store the IGASolid object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new IGA Solids inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all IGA Solids are in

- **func** (function)

Function to call for each IGA Solid

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the IGA Solids in model m:

```
IGASolid.ForEach(m, test);
function test(igso)
{
// igso is IGASolid object
}
```

To call function test for all of the IGA Solids in model m with optional object:

```
var data = { x:0, y:0 };
IGASolid.ForEach(m, test, data);
function test(igso, extra)
{
// igso is IGASolid object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of IGASolid objects or properties for all of the IGA Solids in a model in PRIMER. If the optional property argument is not given then an array of IGASolid objects is returned. If the property argument is given, that property value for each IGA Solid is returned in the array instead of a IGASolid object

Arguments

- **Model** (Model)

Model to get IGA Solids from

- **property (optional)** (string)

Name for property to get for all IGA Solids in the model

Returns

Array of IGASolid objects or properties

Return type

Array

Example

To make an array of IGASolid objects for all of the IGA Solids in model m:

```
var a = IGASolid.GetAll(m);
```

To return an array containing the value of property ’foo’ (for example ’x’ for a node) for each IGA Solid in model m:

```
var a = IGASolid.GetAll(m, ’foo’);
```

GetComments()

Description

Extracts the comments associated to a IGA Solid.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the IGA Solid igso:

```
var comm_array = igso.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of IGASolid objects for all of the flagged IGA Solids in a model in PRIMER If the optional property argument is not given then an array of IGASolid objects is returned. If the property argument is given, then that property value for each IGA Solid is returned in the array instead of a IGASolid object

Arguments

- **Model** (Model)

Model to get IGA Solids from

- **flag** (Flag)

Flag set on the IGA Solids that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged IGA Solids in the model

Returns

Array of IGASolid objects or properties

Return type

Array

Example

To make an array of IGASolid objects for all of the IGA Solids in model m flagged with f:

```
var igso = IGASolid.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the IGA Solids in model m flagged with f:

```
var a = IGASolid.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the IGASolid object for a IGA Solid ID.

Arguments

- **Model** (Model)

Model to find the IGA Solid in

- **number** (integer)

number of the IGA Solid you want the IGASolid object for

Returns

IGASolid object (or null if IGA Solid does not exist).

Return type

IGASolid

Example

To get the IGASolid object for IGA Solid 100 in model m

```
var igso = IGASolid.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a IGASolid property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the IGASolid.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

IGA Solid property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if IGASolid property igso.example is a parameter:

```
Options.property_parameter_names = true;
if (igso.GetParameter(igso.example) ) do_something...
Options.property_parameter_names = false;
```

To check if IGASolid property igso.example is a parameter by using the GetParameter method:

```
if (igso.ViewParameters().GetParameter(igso.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this IGA solid (*IGA_SOLID). **Note that a carriage return is not added**. See also IGASolid.KeywordCards()

Arguments

No arguments

Returns
string containing the keyword.

Return type
String

Example
To get the keyword for IGA solid igs:

```
var key = igs.Keyword();
```

KeywordCards()

Description
Returns the keyword cards for the IGA solid. **Note that a carriage return is not added**. See also IGASolid.Keyword()

Arguments
No arguments

Returns
string containing the cards.

Return type
String

Example
To get the cards for IGA solid igs:

```
var cards = igs.KeywordCards();
```

Last(Model[*Model*]) [static]

Description
Returns the last IGA Solid in the model.

Arguments
- **Model** (Model)

Model to get last IGA Solid in

Returns
IGASolid object (or null if there are no IGA Solids in the model).

Return type
IGASolid

Example
To get the last IGA Solid in model m:

```
var igso = IGASolid.Last(m);
```

Next()

Description

Returns the next IGA Solid in the model.

Arguments

No arguments

Returns

IGASolid object (or null if there are no more IGA Solids in the model).

Return type

IGASolid

Example

To get the IGA Solid in model m after IGA Solid igso:

```
var igso = igso.Next();
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a IGA Solid.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only IGA Solids from that model can be picked. If the argument is a Flag then only IGA Solids that are flagged with *limit* can be selected. If omitted, or null, any IGA Solids from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

IGASolid object (or null if not picked)

Return type

IGASolid

Example

To pick a IGA Solid from model m giving the prompt 'Pick IGA Solid from screen':

```
var igso = IGASolid.Pick('Pick IGA Solid from screen', m);
```

Previous()

Description

Returns the previous IGA Solid in the model.

Arguments

No arguments

Returns

IGASolid object (or null if there are no more IGA Solids in the model).

Return type

IGASolid

Example

To get the IGA Solid in model m before IGA Solid igso:

```
var igso = igso.Previous();
```

RedrawAll(Model*[Model]*) [static]

Description

Redraws all *IGA_SOLIDs whose graphics are out of date in a specified model

Arguments

- **Model** (Model)

Model to redraw IGA SOLIDs in

Returns

No return value

Example

Redraw all out of date *IGA_SOLIDs following a change to an *IGA_SOLID or related IGA entity in model m1:

```
IGASolid.RedrawAll(m1);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select IGA Solids using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting IGA Solids

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only IGA Solids from that model can be selected. If the argument is a Flag then only IGA Solids that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any IGA Solids can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of IGA Solids selected or null if menu cancelled

Return type

Number

Example

To select IGA Solids from model m, flagging those selected with flag f, giving the prompt 'Select IGA Solids':

```
IGASolid.Select(f, 'Select IGA Solids', m);
```

To select IGA Solids, flagging those selected with flag f but limiting selection to IGA Solids flagged with flag l, giving the prompt 'Select IGA Solids':

```
IGASolid.Select(f, 'Select IGA Solids', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the IGA Solid.

Arguments

- **flag** (Flag)

Flag to set on the IGA Solid

Returns

No return value

Example

To set flag f for IGA Solid igso:

```
igso.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the IGA Solid. The IGA Solid will be sketched until you either call IGASolid.Unsketch(), IGASolid.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Solid is sketched. If omitted redraw is true. If you want to sketch several IGA Solids and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch IGA Solid igso:

```
igso.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged IGA Solids in the model. The IGA Solids will be sketched until you either call IGASolid.Unsketch(), IGASolid.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged IGA Solids will be sketched in

- **flag** (Flag)

Flag set on the IGA Solids that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Solids are sketched. If omitted redraw is true. If you want to sketch flagged IGA Solids several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all IGA Solids flagged with flag in model m:

```
IGASolid.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of IGA Solids in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing IGA Solids should be counted. If false or omitted referenced but undefined IGA Solids will also be included in the total.

Returns

number of IGA Solids

Return type

Number

Example

To get the total number of IGA Solids in model m:

```
var total = IGASolid.Total(m);
```

Unblank()

Description

Unblanks the IGA Solid

Arguments

No arguments

Returns

No return value

Example

To unblank IGA Solid igso:

```
igso.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the IGA Solids in the model.

Arguments

- **Model** (Model)

Model that all IGA Solids will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the IGA Solids in model m:

```
IGASolid.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged IGA Solids in the model.

Arguments

- **Model** (Model)

Model that the flagged IGA Solids will be unblanked in

- **flag** (Flag)

Flag set on the IGA Solids that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the IGA Solids in model m flagged with f:

```
IGASolid.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the IGA Solids in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all IGA Solids will be unset in

- **flag** (Flag)

Flag to unset on the IGA Solids

Returns

No return value

Example

To unset the flag f on all the IGA Solids in model m:

```
IGASolid.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the IGA Solid.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Solid is unsketched. If omitted redraw is true. If you want to unsketch several IGA Solids and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch IGA Solid igso:

```
igso.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all IGA Solids.

Arguments

- **Model** (Model)

Model that all IGA Solids will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Solids are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all IGA Solids in model m:

```
IGASolid.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged IGA Solids in the model.

Arguments

- **Model** (Model)

Model that all IGA Solids will be unsketched in

- **flag** (Flag)

Flag set on the IGA Solids that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Solids are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all IGA Solids flagged with flag in model m:

```
IGASolid.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

IGASolid object.

Return type

IGASolid

Example

To check if IGASolid property igso.example is a parameter by using the IGASolid.GetParameter() method:

```
if (igso.ViewParameters().GetParameter(igso.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for IGA Solid with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA Solid object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for IGA Solid igso:

```
igso.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for IGA Solid igso:

```
igso.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for IGA Solid. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for IGA Solid igso:

```
igso.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this IGA Solid.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for IGA Solid igso:

```
var xrefs = igso.Xrefs();
```

---

toString()

Description

Creates a string containing the IGA solid data in keyword format. Note that this contains the keyword header and the keyword cards. See also IGASolid.Keyword() and IGASolid.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for IGA solid igs in keyword format

```
var s = igs.toString();
```

IGATiedEdge class

The IGATiedEdge class gives access to IGA tied edge to edge cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

IGATiedEdge properties

Name	Type	Description
exists (read only)	logical	true if IGA Tied Edge To Edge exists, false if referred to but not defined.
form	integer	Coupling formulation.
id	integer	Apply coupling to entities reference by this ID along topologically connected edges.
include	integer	The Include file number that the IGA Tied Edge To Edge is in.
label (read only)	integer	ID of the IGA tied edge to edge. Only used in PRIMER.

model (read only)	integer	The Model number that the IGA Tied Edge To Edge is in.
sfd	real	Scaling factor for displacement penalty stiffness.
sfr	real	Scaling factor for rotational penalty stiffness.
sft	real	Scaling factor for thin constraint penalty stiffness (rotation free elements).
type	integer	Type of ID

Detailed Description

The IGATiedEdge class allows you to create, modify, edit and manipulate IGA tied edge to edge cards. See the documentation below for more details.

Constructor

new IGATiedEdge(Model*[Model]*, details (optional)*[object]*)

Description

Create a new IGATiedEdge object.

Arguments

- **Model** (Model)

Model that IGA tied edge to edge will be created in.

- **details (optional)** (object)

Details for creating the IGATiedEdge

Object has the following properties:

Name	Type	Description
form (optional)	integer	Coupling formulation.
id (optional)	integer	Apply coupling to entities referenced by the ID field along topologically connected edges.
sfd (optional)	real	Scaling factor for displacement penalty stiffness.
sfr (optional)	real	Scaling factor for rotational penalty stiffness.
sft (optional)	real	Scaling factor for thin constraint penalty stiffness (rotation free elements).
type(optional)	integer	Type of ID.

Returns

IGATiedEdge object

Return type

IGATiedEdge

Example

To create a new IGA tied edge to edge in model m with id 1, type 0, form 0, sfd 2.2, sfr 3.3 and sft 4.4:

```
var igte = new IGATiedEdge(m, {id: 1, type: 0, form: 0, sfd: 2.2, sfr: 3.3, sft: 4.4});
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a IGA Tied Edge To Edge.

Arguments

- **Comment** (Comment)

Comment that will be attached to the IGA Tied Edge To Edge

Returns

No return value

Example

To associate comment c to the IGA Tied Edge To Edge igte:

```
igte.AssociateComment(c);
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse IGA Tied Edge To Edge igte:

```
igte.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the IGA Tied Edge To Edge.

Arguments

- **flag** (Flag)

Flag to clear on the IGA Tied Edge To Edge

Returns

No return value

Example

To clear flag f for IGA Tied Edge To Edge igte:

```
igte.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the IGA Tied Edge To Edge. The target include of the copied IGA Tied Edge To Edge can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

IGATiedEdge object

Return type

IGATiedEdge

Example

To copy IGA Tied Edge To Edge igte into IGA Tied Edge To Edge z:

```
var z = igte.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a IGA Tied Edge To Edge

Arguments

- **Model** (Model)

Model that the IGA Tied Edge To Edge will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

IGATiedEdge object (or null if not made).

Return type

IGATiedEdge

Example

To start creating a IGA Tied Edge To Edge in model m:

```
var igte = IGATiedEdge.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a IGA Tied Edge To Edge.

Arguments

- **Comment** (Comment)

Comment that will be detached from the IGA Tied Edge To Edge

Returns

No return value

Example

To detach comment c from the IGA Tied Edge To Edge igte:

```
igte.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit IGA Tied Edge To Edge igte:

```
igte.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for IGA Tied Edge To Edge with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description

autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA Tied Edge To Edge object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for IGA Tied Edge To Edge igte:

```
igte.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for IGA Tied Edge To Edge igte:

```
igte.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for IGA Tied Edge To Edge. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for IGA Tied Edge To Edge igte:

```
igte.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first IGA Tied Edge To Edge in the model.

Arguments

- **Model** (Model)

Model to get first IGA Tied Edge To Edge in

Returns

IGATiedEdge object (or null if there are no IGA Tied Edge To Edges in the model).

Return type

IGATiedEdge

Example

To get the first IGA Tied Edge To Edge in model m:

```
var igte = IGATiedEdge.First(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the IGA Tied Edge To Edges in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all IGA Tied Edge To Edges will be flagged in

- **flag** (Flag)

Flag to set on the IGA Tied Edge To Edges

Returns

No return value

Example

To flag all of the IGA Tied Edge To Edges with flag f in model m:

```
IGATiedEdge.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the IGA Tied Edge To Edge is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the IGA Tied Edge To Edge

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if IGA Tied Edge To Edge igte has flag f set on it:

```
if (igte.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each IGA Tied Edge To Edge in the model.
Note that ForEach has been designed to make looping over IGA Tied Edge To Edges as fast as possible and so has some limitations.
Firstly, a single temporary IGATiedEdge object is created and on each function call it is updated with the current IGA Tied Edge To Edge data. This means that you should not try to store the IGATiedEdge object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new IGA Tied Edge To Edges inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all IGA Tied Edge To Edges are in

- **func** (function)

Function to call for each IGA Tied Edge To Edge

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the IGA Tied Edge To Edges in model m:

```
IGATiedEdge.ForEach(m, test);
function test(igte)
{
// igte is IGATiedEdge object
}
```

To call function test for all of the IGA Tied Edge To Edges in model m with optional object:

```
var data = { x:0, y:0 };
IGATiedEdge.ForEach(m, test, data);
function test(igte, extra)
{
// igte is IGATiedEdge object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of IGATiedEdge objects or properties for all of the IGA Tied Edge To Edges in a model in PRIMER. If the optional property argument is not given then an array of IGATiedEdge objects is returned. If the property argument is given, that property value for each IGA Tied Edge To Edge is returned in the array instead of a IGATiedEdge object

Arguments

- **Model** (Model)

Model to get IGA Tied Edge To Edges from

- **property (optional)** (string)

Name for property to get for all IGA Tied Edge To Edges in the model

Returns

Array of IGATiedEdge objects or properties

Return type

Array

Example

To make an array of IGATiedEdge objects for all of the IGA Tied Edge To Edges in model m:

```
var a = IGATiedEdge.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each IGA Tied Edge To Edge in model m:

```
var a = IGATiedEdge.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a IGA Tied Edge To Edge.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the IGA Tied Edge To Edge igte:

```
var comm_array = igte.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of IGATiedEdge objects for all of the flagged IGA Tied Edge To Edges in a model in PRIMER If the optional property argument is not given then an array of IGATiedEdge objects is returned. If the property argument is given, then that property value for each IGA Tied Edge To Edge is returned in the array instead of a IGATiedEdge object

Arguments

- **Model** (Model)

Model to get IGA Tied Edge To Edges from

- **flag** (Flag)

Flag set on the IGA Tied Edge To Edges that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged IGA Tied Edge To Edges in the model

Returns

Array of IGATiedEdge objects or properties

Return type

Array

Example

To make an array of IGATiedEdge objects for all of the IGA Tied Edge To Edges in model m flagged with f:

```
var igte = IGATiedEdge.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the IGA Tied Edge To Edges in model m flagged with f:

```
var a = IGATiedEdge.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the IGATiedEdge object for a IGA Tied Edge To Edge ID.

Arguments

- **Model** (Model)

Model to find the IGA Tied Edge To Edge in

- **number** (integer)

number of the IGA Tied Edge To Edge you want the IGATiedEdge object for

Returns

IGATiedEdge object (or null if IGA Tied Edge To Edge does not exist).

Return type

IGATiedEdge

Example

To get the IGATiedEdge object for IGA Tied Edge To Edge 100 in model m

```
var igte = IGATiedEdge.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a IGATiedEdge property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the IGATiedEdge.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

IGA Tied Edge To Edge property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if IGATiedEdge property igte.example is a parameter:

```
Options.property_parameter_names = true;
if (igte.GetParameter(igte.example) ) do_something...
Options.property_parameter_names = false;
```

To check if IGATiedEdge property igte.example is a parameter by using the GetParameter method:

```
if (igte.ViewParameters().GetParameter(igte.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this IGA tied edge to edge (*IGA_TIED_EDGE_TO_EDGE). **Note that a carriage return is not added** See also IGATiedEdge.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To ge the keyword for IGA tied edge to edge igte:

```
var key = igte.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the IGA tied edge to edge. **Note that a carriage return is not added** See also IGATiedEdge.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for IGA tied edge to edge igte:

```
type="JS">var cards = igte.KeywordCards();
```

```
type="python">cards = igte.KeywordCards()
```

Last(Model*[Model]*) [static]

Description

Returns the last IGA Tied Edge To Edge in the model.

Arguments

- **Model** (Model)

Model to get last IGA Tied Edge To Edge in

Returns

IGATiedEdge object (or null if there are no IGA Tied Edge To Edges in the model).

Return type

IGATiedEdge

Example

To get the last IGA Tied Edge To Edge in model m:

```
var igte = IGATiedEdge.Last(m);
```

Next()

Description

Returns the next IGA Tied Edge To Edge in the model.

Arguments

No arguments

Returns

IGATiedEdge object (or null if there are no more IGA Tied Edge To Edges in the model).

Return type

IGATiedEdge

Example

To get the IGA Tied Edge To Edge in model m after IGA Tied Edge To Edge igte:

```
var igte = igte.Next();
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a IGA Tied Edge To Edge.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only IGA Tied Edge To Edges from that model can be picked. If the argument is a Flag then only IGA Tied Edge To Edges that are flagged with *limit* can be selected. If omitted, or null, any IGA Tied Edge To Edges from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

IGATiedEdge object (or null if not picked)

Return type

IGATiedEdge

Example

To pick a IGA Tied Edge To Edge from model m giving the prompt 'Pick IGA Tied Edge To Edge from screen':

```
var igte = IGATiedEdge.Pick('Pick IGA Tied Edge To Edge from screen', m);
```

Previous()

Description

Returns the previous IGA Tied Edge To Edge in the model.

Arguments

No arguments

Returns

IGATiedEdge object (or null if there are no more IGA Tied Edge To Edges in the model).

Return type

IGATiedEdge

Example

To get the IGA Tied Edge To Edge in model m before IGA Tied Edge To Edge igte:

```
var igte = igte.Previous();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select IGA Tied Edge To Edges using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting IGA Tied Edge To Edges

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only IGA Tied Edge To Edges from that model can be selected. If the argument is a Flag then only IGA Tied Edge To Edges that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any IGA Tied Edge To Edges can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of IGA Tied Edge To Edges selected or null if menu cancelled

Return type

Number

Example

To select IGA Tied Edge To Edges from model m, flagging those selected with flag f, giving the prompt 'Select IGA Tied Edge To Edges':

```
IGATiedEdge.Select(f, 'Select IGA Tied Edge To Edges', m);
```

To select IGA Tied Edge To Edges, flagging those selected with flag f but limiting selection to IGA Tied Edge To Edges flagged with flag l, giving the prompt 'Select IGA Tied Edge To Edges':

```
IGATiedEdge.Select(f, 'Select IGA Tied Edge To Edges', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the IGA Tied Edge To Edge.

Arguments

- **flag** (Flag)

Flag to set on the IGA Tied Edge To Edge

Returns

No return value

Example

To set flag f for IGA Tied Edge To Edge igte:

```
igte.SetFlag(f);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of IGA Tied Edge To Edges in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing IGA Tied Edge To Edges should be counted. If false or omitted referenced but undefined IGA Tied Edge To Edges will also be included in the total.

Returns

number of IGA Tied Edge To Edges

Return type

Number

Example

To get the total number of IGA Tied Edge To Edges in model m:

```
var total = IGATiedEdge.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the IGA Tied Edge To Edges in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all IGA Tied Edge To Edges will be unset in

- **flag** (Flag)

Flag to unset on the IGA Tied Edge To Edges

Returns

No return value

Example

To unset the flag f on all the IGA Tied Edge To Edges in model m:

```
IGATiedEdge.UnflagAll(m, f);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

IGATiedEdge object.

Return type

IGATiedEdge

Example

To check if IGATiedEdge property igte.example is a parameter by using the IGATiedEdge.GetParameter() method:

```
if (igte.ViewParameters().GetParameter(igte.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for IGA Tied Edge To Edge with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA Tied Edge To Edge object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for IGA Tied Edge To Edge igte:

```
igte.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for IGA Tied Edge To Edge igte:

```
igte.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for IGA Tied Edge To Edge. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for IGA Tied Edge To Edge igte:

```
igte.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this IGA Tied Edge To Edge.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for IGA Tied Edge To Edge igte:

```
var xrefs = igte.Xrefs();
```

---

toString()

Description

Creates a string containing the IGA tied edge to edge in keyword format. Note that this contains the keyword header and the keyword cards. See also IGATiedEdge.Keyword() and IGATiedEdge.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for IGA tied edge to edge igte in keyword format

```
var s = igte.toString();
```

IGAVolumeXYZ class

The IGAVolumeXYZ class gives you access to IGA volume xyz cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetData(index*[integer]*)
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- RemoveData(index*[integer]*)
- SetData(index*[integer]*, brid*[integer]*)
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

IGAVolumeXYZ constants

Name	Description
IGAVolumeXYZ.BASIS_TRANSFORM	BASIS_TRANSFORM is *IGA_VOLUME_XYZ_BASIS_TRANSFORM.
IGAVolumeXYZ.NONE	NONE is *IGA_VOLUME_XYZ.

IGAVolumeXYZ properties

Name	Type	Description
entries (read only)	integer	Number of Boundary representation IDs (brid) added in IGA_VOLUME_XYZ. Can be used as index to add a new entry.
esid	integer	Parametric edge set ID. *SET_IGA_EDGE_UVW.
exists (read only)	logical	true if IGA Volume XYZ exists, false if referred to but not defined.
fsid	integer	Parametric face set ID. *SET_IGA_FACE_UVW.
include	integer	The Include file number that the IGA Volume XYZ is in.
label	integer	IGAVolumeXYZ number. Also see the vid property which is an alternative name for this.
model (read only)	integer	The Model number that the IGA Volume XYZ is in.
option	constant	IGA Volume XYZ option. • IGAVolumeXYZ.NONE or • IGAVolumeXYZ.BASIS_TRANSFORM
patchid	integer	Physical trivariate NURBS patch ID or a trivariate Basis Transform patch ID.
psid	integer	Parametric point set ID. *SET_IGA_POINT_UVW.
vid	integer	IGAVolumeXYZ number. Also see the label property which is an alternative name for this.

Detailed Description

The IGAVolumeXYZ class allows you to create, modify, edit and manipulate IGA volume xyz cards. See the documentation below for more details.

Constructor

new IGAVolumeXYZ(Model*[Model]*, details*[object]*)

Description

Create a new IGAVolumeXYZ object.

Arguments

- **Model** (Model)

Model that IGA volume xyz will be created in

- **details** (object)

Details for creating the IGAVolumeXYZ

Object has the following properties:

Name	Type	Description
esid (optional)	integer	Parametric edge set ID. *SET_IGA_EDGE_UVW.
fsid (optional)	integer	Parametric face set ID. *SET_IGA_FACE_UVW.

option	constant	IGA Volume XYZ option. IGAVolumeXYZ.NONE or IGAVolumeXYZ.BASIS_TRANSFORM
patchid (optional)	integer	Physical trivariate NURBS patch ID or a trivariate basis transform patch ID.
psid (optional)	integer	Parametric point set ID. *SET_IGA_POINT_UVW.
vid	integer	ID of the IGA Volume XYZ.

Returns

IGAVolumeXYZ object

Return type

IGAVolumeXYZ

Example

To create a new IGA volume xyz in model m using vid 10 with option NONE:

```
var igvx = new IGAVolumeXYZ(m, {option:IGAVolumeXYZ.NONE, vid:10});
```

To create a new IGA volume xyz in model m using vid 21 and option BASIS_TRANSFORM with patch 22, psid 23, esid 24 and fsid 25:

```
var igvx = new IGAVolumeXYZ(m, {option:IGAVolumeXYZ.BASIS_TRANSFORM, vid:21,
patchid: 22, psid:23, esid:24, fsid:25});
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a IGA Volume XYZ.

Arguments

- **Comment** (Comment)

Comment that will be attached to the IGA Volume XYZ

Returns

No return value

Example

To associate comment c to the IGA Volume XYZ igvx:

```
igvx.AssociateComment(c);
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse IGA Volume XYZ igvx:

```
igvx.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the IGA Volume XYZ.

Arguments

- **flag** (Flag)

Flag to clear on the IGA Volume XYZ

Returns

No return value

Example

To clear flag f for IGA Volume XYZ igvx:

```
igvx.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the IGA Volume XYZ. The target include of the copied IGA Volume XYZ can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

IGAVolumeXYZ object

Return type

IGAVolumeXYZ

Example

To copy IGA Volume XYZ igvx into IGA Volume XYZ z:

```
var z = igvx.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a IGA Volume XYZ

Arguments

- **Model** (Model)

Model that the IGA Volume XYZ will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

IGAVolumeXYZ object (or null if not made).

Return type

IGAVolumeXYZ

Example

To start creating a IGA Volume XYZ in model m:

```
var igvx = IGAVolumeXYZ.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a IGA Volume XYZ.

Arguments

- **Comment** (Comment)

Comment that will be detached from the IGA Volume XYZ

Returns

No return value

Example

To detach comment c from the IGA Volume XYZ igvx:

```
igvx.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit IGA Volume XYZ igvx:

```
igvx.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for IGA Volume XYZ with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA Volume XYZ object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for IGA Volume XYZ igvx:

```
igvx.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for IGA Volume XYZ igvx:

```
igvx.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for IGA Volume XYZ. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for IGA Volume XYZ igvx:

```
igvx.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first IGA Volume XYZ in the model.

Arguments

- **Model** (Model)

Model to get first IGA Volume XYZ in

Returns

IGAVolumeXYZ object (or null if there are no IGA Volume XYZs in the model).

Return type

IGAVolumeXYZ

Example

To get the first IGA Volume XYZ in model m:

```
var igvx = IGAVolumeXYZ.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free IGA Volume XYZ label in the model. Also see IGAVolumeXYZ.LastFreeLabel(), IGAVolumeXYZ.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free IGA Volume XYZ label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

IGAVolumeXYZ label.

Return type

Number

Example

To get the first free IGA Volume XYZ label in model m:

```
var label = IGAVolumeXYZ.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the IGA Volume XYZs in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all IGA Volume XYZs will be flagged in

- **flag** (Flag)

Flag to set on the IGA Volume XYZs

Returns

No return value

Example

To flag all of the IGA Volume XYZs with flag f in model m:

```
IGAVolumeXYZ.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the IGA Volume XYZ is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the IGA Volume XYZ

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if IGA Volume XYZ igvx has flag f set on it:

```
if (igvx.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each IGA Volume XYZ in the model.
Note that ForEach has been designed to make looping over IGA Volume XYZs as fast as possible and so has some limitations.
Firstly, a single temporary IGAVolumeXYZ object is created and on each function call it is updated with the current IGA Volume XYZ data. This means that you should not try to store the IGAVolumeXYZ object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new IGA Volume XYZs inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all IGA Volume XYZs are in

- **func** (function)

Function to call for each IGA Volume XYZ

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the IGA Volume XYZs in model m:

```
IGAVolumeXYZ.ForEach(m, test);
function test(igvx)
{
// igvx is IGAVolumeXYZ object
}
```

To call function test for all of the IGA Volume XYZs in model m with optional object:

```
var data = { x:0, y:0 };
IGAVolumeXYZ.ForEach(m, test, data);
function test(igvx, extra)
{
// igvx is IGAVolumeXYZ object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of IGAVolumeXYZ objects or properties for all of the IGA Volume XYZs in a model in PRIMER. If the optional property argument is not given then an array of IGAVolumeXYZ objects is returned. If the property argument is given, that property value for each IGA Volume XYZ is returned in the array instead of a IGAVolumeXYZ object

Arguments

- **Model** (Model)

Model to get IGA Volume XYZs from

- **property (optional)** (string)

Name for property to get for all IGA Volume XYZs in the model

Returns

Array of IGAVolumeXYZ objects or properties

Return type

Array

Example

To make an array of IGAVolumeXYZ objects for all of the IGA Volume XYZs in model m:

```
var a = IGAVolumeXYZ.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each IGA Volume XYZ in model m:

```
var a = IGAVolumeXYZ.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a IGA Volume XYZ.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the IGA Volume XYZ igvx:

```
var comm_array = igvx.GetComments();
```

GetData(index*[integer]*)

Description

Returns the data for brid in *IGA_VOLUME_XYZ. Only valid for option IGAVolumeXYZ.NONE.

Arguments

- **index** (integer)

Index you want the data for. **Note that indices start at 0**.

Returns

The ID of 2 dimensional boundary representation.

Return type

Number

Example

To get the 3rd data entry in IGA volume xyz igvx:

```
var data = igvx.GetData(2);
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of IGAVolumeXYZ objects for all of the flagged IGA Volume XYZs in a model in PRIMER If the optional property argument is not given then an array of IGAVolumeXYZ objects is returned. If the property argument is given, then that property value for each IGA Volume XYZ is returned in the array instead of a IGAVolumeXYZ object

Arguments

- **Model** (Model)

Model to get IGA Volume XYZs from

- **flag** (Flag)

Flag set on the IGA Volume XYZs that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged IGA Volume XYZs in the model

Returns

Array of IGAVolumeXYZ objects or properties

Return type

Array

Example

To make an array of IGAVolumeXYZ objects for all of the IGA Volume XYZs in model m flagged with f:

```
var igvx = IGAVolumeXYZ.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the IGA Volume XYZs in model m flagged with f:

```
var a = IGAVolumeXYZ.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the IGAVolumeXYZ object for a IGA Volume XYZ ID.

Arguments

- **Model** (Model)

Model to find the IGA Volume XYZ in

- **number** (integer)

number of the IGA Volume XYZ you want the IGAVolumeXYZ object for

Returns

IGAVolumeXYZ object (or null if IGA Volume XYZ does not exist).

Return type

IGAVolumeXYZ

Example

To get the IGAVolumeXYZ object for IGA Volume XYZ 100 in model m

```
var igvx = IGAVolumeXYZ.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a IGAVolumeXYZ property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the IGAVolumeXYZ.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

IGA Volume XYZ property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if IGAVolumeXYZ property igvx.example is a parameter:

```
Options.property_parameter_names = true;
if (igvx.GetParameter(igvx.example) ) do_something...
Options.property_parameter_names = false;
```

To check if IGAVolumeXYZ property igvx.example is a parameter by using the GetParameter method:

```
if (igvx.ViewParameters().GetParameter(igvx.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this IGA volume xyz (*IGA_VOLUME_XYZ). **Note that a carriage return is not added**. See also IGAVolumeXYZ.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for IGA volume xyz igvx:

```
var key = igvx.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the IGA volume xyz. **Note that a carriage return is not added**. See also IGAVolumeXYZ.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for IGA volume xyz igvx:

```
var cards = igvx.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last IGA Volume XYZ in the model.

Arguments

- **Model** (Model)

Model to get last IGA Volume XYZ in

Returns

IGAVolumeXYZ object (or null if there are no IGA Volume XYZs in the model).

Return type

IGAVolumeXYZ

Example

To get the last IGA Volume XYZ in model m:

```
var igvx = IGAVolumeXYZ.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free IGA Volume XYZ label in the model. Also see IGAVolumeXYZ.FirstFreeLabel(), IGAVolumeXYZ.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free IGA Volume XYZ label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

IGAVolumeXYZ label.

Return type

Number

Example

To get the last free IGA Volume XYZ label in model m:

```
var label = IGAVolumeXYZ.LastFreeLabel(m);
```

Next()

Description

Returns the next IGA Volume XYZ in the model.

Arguments

No arguments

Returns

IGAVolumeXYZ object (or null if there are no more IGA Volume XYZs in the model).

Return type

IGAVolumeXYZ

Example

To get the IGA Volume XYZ in model m after IGA Volume XYZ igvx:

```
var igvx = igvx.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) IGA Volume XYZ label in the model. Also see IGAVolumeXYZ.FirstFreeLabel(), IGAVolumeXYZ.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free IGA Volume XYZ label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

IGAVolumeXYZ label.

Return type

Number

Example

To get the next free IGA Volume XYZ label in model m:

```
var label = IGAVolumeXYZ.NextFreeLabel(m);
```

Previous()

Description

Returns the previous IGA Volume XYZ in the model.

Arguments

No arguments

Returns

IGAVolumeXYZ object (or null if there are no more IGA Volume XYZs in the model).

Return type

IGAVolumeXYZ

Example

To get the IGA Volume XYZ in model m before IGA Volume XYZ igvx:

```
var igvx = igvx.Previous();
```

RemoveData(index*[integer]*)

Description

Removes brid for an index in *IGA_VOLUME_XYZ. Only valid for option IGAVolumeXYZ.NONE.

Arguments

- **index** (integer)

The index you want to delete brid for. **Note that indices start at 0, not 1**.

Returns

No return value.

Example

To delete the brid for the 3rd index for *IGA_VOLUME_XYZ in model m:

```
if (igvx.entries >= 3)
{
    igvx.RemoveData(2);
}
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the IGA Volume XYZs in the model.

Arguments

- **Model** (Model)

Model that all IGA Volume XYZs will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the IGA Volume XYZs in model m, from 1000000:

```
IGAVolumeXYZ.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged IGA Volume XYZs in the model.

Arguments

- **Model** (Model)

Model that all the flagged IGA Volume XYZs will be renumbered in

- **flag** (Flag)

Flag set on the IGA Volume XYZs that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the IGA Volume XYZs in model m flagged with f, from 1000000:

```
IGAVolumeXYZ.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select IGA Volume XYZs using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting IGA Volume XYZs

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only IGA Volume XYZs from that model can be selected. If the argument is a Flag then only IGA Volume XYZs that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any IGA Volume XYZs can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of IGA Volume XYZs selected or null if menu cancelled

Return type

Number

Example

To select IGA Volume XYZs from model m, flagging those selected with flag f, giving the prompt 'Select IGA Volume XYZs':

```
IGAVolumeXYZ.Select(f, 'Select IGA Volume XYZs', m);
```

To select IGA Volume XYZs, flagging those selected with flag f but limiting selection to IGA Volume XYZs flagged with flag l, giving the prompt 'Select IGA Volume XYZs':

```
IGAVolumeXYZ.Select(f, 'Select IGA Volume XYZs', l);
```

SetData(index*[integer]*, brid*[integer]*)

Description

Sets brid for *IGA_VOLUME_XYZ. Only valid for option IGAVolumeXYZ.NONE.

Arguments

- **index** (integer)

Index you want to set the brid for. **Note that indices start at 0**.

- **brid** (integer)

The ID of 2 dimensional boundary representation.

Returns

No return value.

Example

To add an entry to the *IGA_VOLUME_XYZ igvx with id 10 by using "entries" property to get the next available index:

```
var i = igvx.entries;
igvx.SetData(i, 10);
```

To change an entry in 3rd field for *IGA_VOLUME_XYZ igvx with id 30:

```
igvx.SetData(2, 30);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the IGA Volume XYZ.

Arguments

- **flag** (Flag)

Flag to set on the IGA Volume XYZ

Returns

No return value

Example

To set flag f for IGA Volume XYZ igvx:

```
igvx.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the IGA Volume XYZ. The IGA Volume XYZ will be sketched until you either call IGAVolumeXYZ.Unsketch(), IGAVolumeXYZ.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Volume XYZ is sketched. If omitted redraw is true. If you want to sketch several IGA Volume XYZs and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch IGA Volume XYZ igvx:

```
igvx.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged IGA Volume XYZs in the model. The IGA Volume XYZs will be sketched until you either call IGAVolumeXYZ.Unsketch(), IGAVolumeXYZ.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged IGA Volume XYZs will be sketched in

- **flag** (Flag)

Flag set on the IGA Volume XYZs that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Volume XYZs are sketched. If omitted redraw is true. If you want to sketch flagged IGA Volume XYZs several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all IGA Volume XYZs flagged with flag in model m:

```
IGAVolumeXYZ.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of IGA Volume XYZs in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing IGA Volume XYZs should be counted. If false or omitted referenced but undefined IGA Volume XYZs will also be included in the total.

Returns

number of IGA Volume XYZs

Return type

Number

Example

To get the total number of IGA Volume XYZs in model m:

```
var total = IGAVolumeXYZ.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the IGA Volume XYZs in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all IGA Volume XYZs will be unset in

- **flag** (Flag)

Flag to unset on the IGA Volume XYZs

Returns

No return value

Example

To unset the flag f on all the IGA Volume XYZs in model m:

```
IGAVolumeXYZ.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the IGA Volume XYZ.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Volume XYZ is unsketched. If omitted redraw is true. If you want to unsketch several IGA Volume XYZs and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch IGA Volume XYZ igvx:

```
igvx.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all IGA Volume XYZs.

Arguments

- **Model** (Model)

Model that all IGA Volume XYZs will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Volume XYZs are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all IGA Volume XYZs in model m:

```
IGAVolumeXYZ.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged IGA Volume XYZs in the model.

Arguments

- **Model** (Model)

Model that all IGA Volume XYZs will be unsketched in

- **flag** (Flag)

Flag set on the IGA Volume XYZs that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the IGA Volume XYZs are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all IGA Volume XYZs flagged with flag in model m:

```
IGAVolumeXYZ.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

IGAVolumeXYZ object.

Return type

IGAVolumeXYZ

Example

To check if IGAVolumeXYZ property igvx.example is a parameter by using the IGAVolumeXYZ.GetParameter() method:

```
if (igvx.ViewParameters().GetParameter(igvx.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for IGA Volume XYZ with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The IGA Volume XYZ object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for IGA Volume XYZ igvx:

```
igvx.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for IGA Volume XYZ igvx:

```
igvx.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for IGA Volume XYZ. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for IGA Volume XYZ igvx:

```
igvx.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this IGA Volume XYZ.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for IGA Volume XYZ igvx:

```
var xrefs = igvx.Xrefs();
```

---

toString()

Description

Creates a string containing the IGA volume xyz data in keyword format. Note that this contains the keyword header and the keyword cards. See also IGAVolumeXYZ.Keyword() and IGAVolumeXYZ.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for IGA volume xyz igvx in keyword format

```
var s = igvx.toString();
```

Include class

The Include class allows you to access the include files in a model. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*, masterInclude (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- First(Model*[Model]*)
- FlagAll(Model*[Model]*, flag*[Flag]*, masterInclude (optional)*[boolean]*)
- GetAll(Model*[Model]*, masterInclude (optional)*[boolean]*)
- GetFromID(Model*[Model]*, include number*[integer]*)
- Last(Model*[Model]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- Select(flag*[Flag]*, prompt*[string]*, Model (optional)*[Model]*, modal (optional)*[boolean]*)
- Total(Model*[Model]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)

Member functions

- ClearFlag(flag*[Flag]*, clear contents (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetDetailedRange(type argument*[string]*)
- GetLockedLabelData(rangenum*[integer]*)
- IsEmpty()
- Keyword()
- KeywordCards()
- MakeCurrentLayer()
- Modified(listing*[boolean]*)
- Next()
- Previous()
- RemoveLockedLabelData(rangenum*[integer]*)
- SetDetailedRange(type argument*[string]*, min label*[integer]*, max label*[integer]*)
- SetFlag(flag*[Flag]*, flag contents (optional)*[boolean]*)
- SetLockedLabelData(rangenum*[integer]*, min*[integer]*, max*[integer]*, type*[string]*, safe (optional)*[boolean]*, all_includes (optional)*[boolean]*)
- SetTransformOffset(offset*[constant]*, value*[integer]*, check_only (optional)*[boolean]*)
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Write(filename*[string]*, options (optional)*[object]*)
- toString()

Include constants

Name	Description
Include.COPY_INTO_CURRENT	Copied elements are put into the current layer. See also Options.copy_target_include
Include.COPY_INTO_SOURCE	Copied elements are put into the include of the original element. See also Options.copy_target_include

Include.MASTER_ONLY	Only write the master file. See also Model.Write()
Include.MERGE	Merge include files into the master file. See also Model.Write()
Include.NOT_WRITTEN	Prevent include files from being written. See also Model.Write()
Include.SAME_DIR	Write master and include files into the same directory. See also Model.Write()
Include.SELECT	Select include files to be written out. See also Model.Write()
Include.SUBDIR	Write include files to subdirectory. See also Model.Write()

Constants for Directory separators

Name	Description
Include.NATIVE	Use directory separators native to this machine when writing directory names. See also Model.Write()
Include.UNIX	Use unix directory separators when writing directory names. See also Model.Write()
Include.WINDOWS	Use windows directory separators when writing directory names. See also Model.Write()

Constants for Pathnames

Name	Description
Include.ABSOLUTE	Write include file with absolute pathname. See also Model.Write()
Include.RELATIVE	Write include file with relative pathname. See also Model.Write()

Constants for Transformation offsets

Name	Description
Include.ENDOFF	Offset applied to PRIMER post end keywords (Dummy, Mechanism etc.)
Include.IDDOFF	Offset to define ID (used in Include.SetTransformOffset())
Include.IDEOFF	Offset to element ID (used in Include.SetTransformOffset())
Include.IDFOFF	Offset to function and table ID (used in Include.SetTransformOffset())
Include.IDMOFF	Offset to material ID (used in Include.SetTransformOffset())
Include.IDNOFF	Offset to node ID (used in Include.SetTransformOffset())
Include.IDPOFF	Offset to part ID (used in Include.SetTransformOffset())
Include.IDROFF	Offset to other ID (used in Include.SetTransformOffset())
Include.IDSOFF	Offset to set ID (used in Include.SetTransformOffset())

Constants for compress mode

Name	Description
Include.INDIVIDUAL_GZIP	Each file ’name.key’ is ’gzipped’ to become the individual file ’name.key.gz’
Include.INDIVIDUAL_ZIP	Each file ’name.key’ is ’zipped’ to become the individual file ’name.key.zip’
Include.KEEP_ORIGINAL	Each file ’name.key’ is written using its original compression: uncompressed, ’.gz. or ’.zip’ format

Include properties

Name	Type	Description
comments	string	Comments stored at the top of the include file. Note that this property is not supported for master include file.
fctchg	real	Electric charge transformation factor. Note that this property is not supported for master include file.
fctlen	real	Length transformation factor. Note that this property is not supported for master include file.
fctmas	real	Mass transformation factor. Note that this property is not supported for master include file.
fcttem	string	Temperature transformation factor. Note that this property is not supported for master include file.
fcttim	real	Time transformation factor. Note that this property is not supported for master include file.
file	string	The absolute filename for this include file.Note that this property is not supported for master include file. Also see the name and path properties.
genmax	integer	Include maximum label range value for general items
genmin	integer	Include minimum label range value for general items
iddoff (read only)	integer	Offset to define ID. To set property use Include.SetTransformOffset(). Note that this property is not supported for master include file.
ideoff (read only)	integer	Offset to element ID. To set property use Include.SetTransformOffset(). Note that this property is not supported for master include file.
idfoff (read only)	integer	Offset to function and table ID. To set property use Include.SetTransformOffset(). Note that this property is not supported for master include file.
idmoff (read only)	integer	Offset to material and equation of state ID. To set property use Include.SetTransformOffset(). Note that this property is not supported for master include file.
idnoff (read only)	integer	Offset to node ID. To set property use Include.SetTransformOffset(). Note that this property is not supported for master include file.
idpoff (read only)	integer	Offset to part, nodal rigid body and constrained node set ID. To set property use Include.SetTransformOffset(). Note that this property is not supported for master include file.
idroff (read only)	integer	Offset to other ID. To set property use Include.SetTransformOffset(). Note that this property is not supported for master include file.
idsoff (read only)	integer	Offset to set ID. To set property use Include.SetTransformOffset(). Note that this property is not supported for master include file.
incout	integer	Create file containing transformed data. Note that this property is not supported for master include file.
label (read only)	integer	Include number. This number is used to identify the include file. A number is required as it is possible (with include transforms) to have multiple include files with the same name so they cannot be identified by name. The master file is include file number 0. Also see the parent property.
model	integer	The Model number that the include is in.
n_locked_range	integer	Number of locked label ranges. Note that this does not include label ranges locked model-wide (ALL includes).
name	string	The filename for this include file excluding any path. Note that this property is not supported for master include file. Also see the file and path properties.
nelmax	integer	Include maximum label range value for nodes/elements/nrbc/const. spotwelds/define HWA items
nelmin	integer	Include minimum label range value for nodes/elements/nrbc/const. spotwelds/define HWA items
parent	integer	Include number for the parent include file of this include. This number is used to identify the parent include file. A number is required as it is possible (with include transforms) to have multiple include files with the same name so they cannot be identified by name. The master file is include file number 0. Also see the label property. Note that this property is not supported for master include file.

path	string	The path for this include file. Note that this property is not supported for master include file. Also see the file and name properties.
suppressed	logical	If keyout of Include file has been suppressed. Note that this property is not supported for master include file.
tranid	integer	Define transformation number. Note that this property is not supported for master include file.
transform	logical	true if this include file is an *INCLUDE_TRANSFORM, false otherwise. Note that this property is not supported for master include file.

Detailed Description

The Include class allows to create and query include files in a model. See the documentation below for more details.

Constructor

new Include(Model*[Model]*, name*[string]*, parent (optional)*[integer]*)

Description

Create a new Include object.

Arguments

- **Model** (Model)

Model that include will be created in

- **name** (string)

Include filename

- **parent (optional)** (integer)

Parent include file number. If omitted parent will be 0 (main file).

Returns

Include object

Return type

Include

Example

To create a new include file /path/to/include.key in model m

```
var i = new Include(m, "/path/to/include.key");
```

Details of functions

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*, masterInclude (optional)*[boolean]*) [static]

Description

Blanks all of the includes in the model.

Arguments

- **Model** (Model)

Model that all includes will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

- **masterInclude (optional)** (boolean)

If masterInclude file should be blanked or not. If omitted masterInclude is false. The master file is include file number 0.

Returns

No return value

Example

To blank all of the includes in model m:

```
Include.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged include files in the model.

Arguments

- **Model** (Model)

Model that all the flagged includes will be blanked in

- **flag** (Flag)

Flag set on the includes that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the include files in model m flagged with f:

```
Include.BlankFlagged(m, f);
```

ClearFlag(flag*[Flag]*, clear contents (optional)*[boolean]*)

Description

Clears a flag on the include.

Arguments

- **flag** (Flag)

Flag to clear on the include

- **clear contents (optional)** (boolean)

If true then the items in the include file will also have flag cleared. If false (default) then the include file contents are not cleared.

Returns

Number of item flags cleared

Return type

Number

Example

To clear flag f for include i:

```
i.ClearFlag(f);
```

To clear flag f for include i and all of the items inside the include file, returning the number of item flags cleared in the include file:

```
var ncleared = i.ClearFlag(f, true);
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for an include file with an optional custom autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply a custom autofix. If omitted no autofix will be possible (default). The include object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, a custom autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe the custom autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for include i:

```
i.Error("My custom error");
```

To add an error message "My custom error" with a custom autofix "My custom autofix", calling function my_autofix, for include i:

```
i.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for an include file. For more details on checking see the Check class. Note that this function is not supported for the master include file.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for include i:

```
i.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first include file in the model.

Arguments

- **Model** (Model)

Model to get first include in

Returns

Include object (or null if there are no includes in the model).

Return type

Include

Example

To get the first include in model m:

```
var i = Include.First(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*, masterInclude (optional)*[boolean]*) [static]

Description

Flags all of the includes in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all includes will be flagged in

- **flag** (Flag)

Flag to set on the includes

- **masterInclude (optional)** (boolean)

If masterInclude file should be flagged or not. If omitted masterInclude is false. The master file is include file number 0.

Returns

No return value

Example

To flag all of the includes with flag f in model m:

```
Include.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the include is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the include

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if include i has flag f set on it:

```
if (i.Flagged(f) ) do_something...
```

GetAll(Model*[Model]*, masterInclude (optional)*[boolean]*) [static]

Description

Returns an array of Include objects for all of the includes in a model in PRIMER

Arguments

- **Model** (Model)

Model to get includes from

- **masterInclude (optional)** (boolean)

If masterInclude file should be included or not. If omitted masterInclude is false. The master file is include file number 0.

Returns

Array of Include objects

Return type

Array

Example

To make an array of Include objects for all of the includes in model m

```
var i = Include.GetAll(m);
```

GetDetailedRange(type argument*[string]*)

Description

Gets detailed min and max label ranges for specified type from the include.

Arguments

- **type argument** (string)

Entity type for which ranges are returned

Returns

An array containing the min and max label ranges for the specified type or null if no range defined for this type.

Return type

Array

Example

To get node ranges for include i:

```
var ranges = i.GetDetailedRange("NODE");
var min = ranges[0];
var max = ranges[1];
```

GetFromID(Model*[Model]*, include number*[integer]*) [static]

Description

Returns the Include object for an include label.
Note that items that are in the main keyword file will have a layer value of 0 which can be used as the *include number* argument to this function to return master include file.

Arguments

- **Model** (Model)

Model to find the include in

- **include number** (integer)

number of the include you want the Include object for

Returns

Include object (or null if include does not exist).

Return type

Include

Example

To get the Include object for include 10 in model m

```
var i = Include.GetFromID(m, 10);
```

GetLockedLabelData(rangenum*[integer]*)

Description

Returns the locked label data for include files. Also see the n_locked_range property

Arguments

- **rangenum** (integer)

The range number you want the data for; includes can have multiple ranges. **Note that range numbers start at 0, not 1**.

Returns

An array containing the include name (string can also be "ALL" if range is applicable model-wide), start (min) label (integer), end (max) label (integer), safe range (0 or 1 for false or true), and entity type (string).

Return type

Number

Example

To get the locked label data for the 3rd range for include i:

```
if (i.n_locked_range >= 3)
{
    var locked_label_data = i.GetLockedLabelData(2);
}
```

IsEmpty()

Description

Returns true if include is Empty (contains no INSTALLED static/sort/kid/include items).

Arguments

No arguments

Returns

logical

Return type

Boolean

Example

To see if include inc is empty

```
if(inc.Empty())
```

Keyword()

Description

Returns the keyword for this include (*INCLUDE, *INCLUDE_TRANSFORM). **Note that a carriage return is not added**. See also Include.KeywordCards(). This function is not supported for the master include file.

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for include i:

```
var key = i.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the include. **Note that a carriage return is not added**. See also Include.Keyword(). Also note that this function is not supported for the master include file.

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for include i:

```
var cards = i.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last include file in the model.

Arguments

- **Model** (Model)

Model to get last include in

Returns

Include object (or null if there are no includes in the model).

Return type

Include

Example

To get the last include in model m:

```
var i = Include.Last(m);
```

MakeCurrentLayer()

Description

Sets this include file to be the current layer so that any newly created items are put in this include file. Also see the Model.layer property.

Arguments

No arguments

Returns

No return value

Example

To make include i the current layer:

```
i.MakeCurrentLayer();
```

Modified(listing*[boolean]*)

Description

Returns true if include has been modified.

Arguments

- **listing** (boolean)

false for no listing output, true for listing output

Returns

logical

Return type

Boolean

Example

To see if include inc is modified

```
if(inc.Modified(false)) ... (no listing output)
```

Next()

Description

Returns the next include in the model. Note that this function is not supported for the master include file.

Arguments

No arguments

Returns

Include object (or null if there are no more includes in the model).

Return type

Include

Example

To get the include in model m after include i:

```
var i = i.Next();
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick an include.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only includes from that model can be picked. If the argument is a Flag then only includes that are flagged with *limit* can be selected. If omitted, or null, any includes from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

Include object (or null if not picked)

Return type

Include

Example

To pick an includee from model m giving the prompt 'Pick include from screen':

```
var i = Include.Pick('Pick include from screen', m);
```

Previous()

Description

Returns the previous include in the model. Note that this function is not supported for the master include file.

Arguments

No arguments

Returns

Include object (or null if there are no more includes in the model).

Return type

Include

Example

To get the include in model m before include i:

```
var i = i.Previous();
```

RemoveLockedLabelData(rangenum*[integer]*)

Description

Removes the locked label data for a range in include files. Also see the n_locked_range property

Arguments

- **rangenum** (integer)

The locked label range you want to remove. **Note that range numbers start at 0, not 1**.

Returns

No return value.

Example

To remove the locked labels for the 3rd range for include i:

```
i.RemoveLockedLabelData(2);
```

Select(flag*[Flag]*, prompt*[string]*, Model (optional)*[Model]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select includes using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting includes

- **prompt** (string)

Text to display as a prompt to the user

- **Model (optional)** (Model)

Model to select from

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of items selected or null if menu cancelled

Return type

Number

Example

To select an include from model m, flagging those selected with flag f, giving the prompt 'Select include':

```
Include.Select(f, 'Select include', m);
```

SetDetailedRange(type argument*[string]*, min label*[integer]*, max label*[integer]*)

Description

Sets detailed min and max label ranges for specified type on the include.

Arguments

- **type argument** (string)

Entity type for which ranges are to be defined

- **min label** (integer)

Defines the smallest label for entities of this type

- **max label** (integer)

Defines the largest label for entities of this type

Returns

No return value

Example

To set node ranges for include i:

```
i.SetDetailedRange("NODE", 50000, 60000);
```

SetFlag(flag*[Flag]*, flag contents (optional)*[boolean]*)

Description

Sets a flag on the include.

Arguments

- **flag** (Flag)

Flag to set on the include

- **flag contents (optional)** (boolean)

If true then the items in the include file will also be flagged. If false (default) then the include file contents are not flagged.

Returns

Number of items flagged

Return type

Number

Example

To set flag f for include i:

```
i.SetFlag(f);
```

To set flag f for include i and all of the items inside the include file, returning the number of items flagged in the include file:

```
var nflagged = i.SetFlag(f, true);
```

SetLockedLabelData(rangenum*[integer]*, min*[integer]*, max*[integer]*, type*[string]*, safe (optional)*[boolean]*, all_includes (optional)*[boolean]*)

Description

Sets the locked label data for a particular range for an include file. Also see the n_locked_range property

Arguments

- **rangenum** (integer)

The range you want to set the data for. **Note that range numbers start at 0, not 1**.

- **min** (integer)

Start (min) label for a locked range.

- **max** (integer)

End (max) label for a locked range.

- **type** (string)

Entity type code - "NODE", "SHELL" etc. Can also be "ALL" (for a list of types see Appendix I of the PRIMER manual).

- **safe (optional)** (boolean)

Determines whether a locked range is safe (protected).

- **all_includes (optional)** (boolean)

Specified range will be set model-wide (all includes). Only useful when working with the 'master' include.

Returns

No return value.

Example

To set the locked label data for the 3rd range with min 99, max 199, for nodes for include i:

```
i.SetLockedLabelData(3, 99, 199, "NODE");
```

SetTransformOffset(offset*[constant]*, value*[integer]*, check_only (optional)*[boolean]*)

Description

Sets offset values for include transform. This function is required to change the offset values rather than changing the properties directly so that the include can be checked to ensure that the new value does not cause any label clashes with existing items or any negative labels when the transform is unapplied when writing the include. Note that this function is not supported for the master include file.

Arguments

- **offset** (constant)

The include transform offset type to change. Can be Include.IDNOFF, Include.IDEOFF, Include.IDPOFF, Include.IDMOFF, Include.IDSOFF, Include.IDFOFF, Include.IDDOFF or Include.IDROFF.

- **value** (integer)

The value to change the offset to

- **check_only (optional)** (boolean)

Sometimes it may be necessary to check if changing an offset for an include will cause an error or label clash rather than actually changing it. If check only is true then PRIMER will just check to see if the new value for the offset will cause any label clashes or negative labels **and not change the offset value or any item labels**. If false or omitted then the offset and labels will be updated if there are no errors.

Returns

logical, true if change successful. false if the change would cause a clash of labels or negative labels, in which case the value is not changed.

Return type

Boolean

Example

To set idpoff for include i to 1000, checking that the change is successful:

```
var success = i.SetTransformOffset(Include.IDPOFF, 1000);
```

Total(Model*[Model]*) [static]

Description

Returns the total number of include files in the model.

Arguments

- **Model** (Model)

Model to get include total from

Returns

integer

Return type

Number

Example

To get the number of include files in model m:

```
var t = Include.Total(m);
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the includes in the model.

Arguments

- **Model** (Model)

Model that all includes will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the includes in model m:

```
Include.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged include files in the model.

Arguments

- **Model** (Model)

Model that all the flagged includes will be unblanked in

- **flag** (Flag)

Flag set on the includes that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the include files in model m flagged with f:

```
Include.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the includes in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all includes will be unset in

- **flag** (Flag)

Flag to unset on the includes

Returns

No return value

Example

To unset the flag f on all of the includes in model m:

```
Include.UnflagAll(m, f);
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for an include file with an optional custom autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description

autofixFunction (optional)	function	An optional function to call to apply a custom autofix. If omitted no autofix will be possible (default). The include object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, a custom autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe the custom autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for include i:

```
i.Warning("My custom warning");
```

To add a warning message "My custom warning" with a custom autofix "My custom autofix", calling function my_autofix, for include i:

```
i.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for an include file. For more details on checking see the Check class. Note that this function is not supported for the master include file.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for include i:

```
i.Warning("My custom warning");
```

Write(filename*[string]*, options (optional)*[object]*)

Description

Writes an include file. Note that this function is not supported for the master include file.

Arguments

- **filename** (string)

Filename of the Ansys LS-DYNA keyword file you want to write

- **options (optional)** (object)

Options specifying how the file should be written out. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
binary (optional)	boolean	If true then the output file will be written out in binary. If false (default) then an ascii file will be written.
compress (optional)	boolean	If true then the output file will be compressed. If false (default) then an uncompressed file will be written.
compressLevel (optional)	integer	Compression level for .gz and .zip files. Must be in the range 1 to 9 with 1 being the least compression (fastest speed) to 9 being the greatest compression (slowest speed)
compressMode (optional)	integer	This option can be used to specify the mode of compression. Can be Include.KEEP_ORIGINAL or Include.INDIVIDUAL_GZIP or Include.INDIVIDUAL_ZIP
fileStartAscii (optional)	boolean	If true then the begining of the file (*CONTROL etc) file is written out in ascii. If false (default) then the entire file is converted to binary.
i10 (optional)	boolean	If true then i10 format will be used to write the file. If false (default) then the normal Ansys LS-DYNA format will be used.
large (optional)	boolean	If true then large format will be used to write the file. If false (default) then the normal Ansys LS-DYNA format will be used. Note that large format is only available from version R7.1 and above.
path (optional)	integer	The method used to write include paths. Can be Include.ABSOLUTE (default) or Include.RELATIVE
separator (optional)	integer	The directory separator used when writing include files. Can be Include.NATIVE (default), Include.UNIX or Include.WINDOWS
version (optional)	string	The Ansys LS-DYNA version used to write the file. Can be "971R5", "971R4", "971R3", "970v6763" etc (see the version popup in Model->Write '>>> Ansys LS-DYNA output options' for a full list). See also Options.dyna_version

Returns

No return value

Example

To Write include i to file /data/test/file.key as a compressed gzip in version R10.0

```
var output_obj = new Object();
output_obj.version = "R10.0";
output_obj.compress = true;
output_obj.compressMode = Include.INDIVIDUAL_GZIP;
i.Write("/data/test/file.key", output_obj);
```

toString()

Description

Creates a string containing the include data in keyword format. Note that this contains the keyword header and the keyword cards. See also Include.Keyword() and Include.KeywordCards(). Also note that this function is not supported for the master include file.

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for include i in keyword format

```
var s = i.toString();
```

AxialForceBeam class

The AxialForceBeam class gives you access to initial axial force beam cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- First(Model*[Model]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

AxialForceBeam properties

Name	Type	Description
bsid	integer	Beam set ID.
exists (read only)	logical	true if axial force beam exists, false if referred to but not defined.
id (read only)	integer	ID of the axial force beam. Only used in PRIMER.
include	integer	The Include file number that the axial force beam is in.

kbend	integer	Bending stiffness flag.
lcid	integer	Loadcurve ID.
model (read only)	integer	The Model number that the axial force beam is in.
scale	real	Scale factor on loadcurve.

Detailed Description

The AxialForceBeam class allows you to create, modify, edit and manipulate initial axial force beam cards. See the documentation below for more details.

Constructor

new AxialForceBeam(Model*[Model]*, bsid*[integer]*, lcid*[integer]*, scale (optional)*[real]*)

Description

Create a new AxialForceBeam object.

Arguments

- **Model** (Model)

Model that axial force beam will be created in

- **bsid** (integer)

BeamSet ID.

- **lcid** (integer)

Loadcurve ID defining preload versus time.

- **scale (optional)** (real)

Scale factor on curve

Returns

AxialForceBeam object

Return type

AxialForceBeam

Example

To create a new axial force beam in model m using beam set 10, load curve 100:

```
var afb = new AxialForceBeam(m, 10, 100);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a axial force beam.

Arguments

- **Comment** (Comment)

Comment that will be attached to the axial force beam

Returns

No return value

Example

To associate comment c to the axial force beam afb:

```
afb.AssociateComment(c);
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the axial force beam.

Arguments

- **flag** (Flag)

Flag to clear on the axial force beam

Returns

No return value

Example

To clear flag f for axial force beam afb:

```
afb.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the axial force beam. The target include of the copied axial force beam can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

AxialForceBeam object

Return type

AxialForceBeam

Example

To copy axial force beam afb into axial force beam z:

```
var z = afb.Copy();
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a axial force beam.

Arguments

- **Comment** (Comment)

Comment that will be detached from the axial force beam

Returns

No return value

Example

To detach comment c from the axial force beam afb:

```
afb.DetachComment(c);
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for axial force beam with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The axial force beam object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for axial force beam afb:

```
afb.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for axial force beam afb:

```
afb.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for axial force beam. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for axial force beam afb:

```
afb.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first axial force beam in the model.

Arguments

- **Model** (Model)

Model to get first axial force beam in

Returns

AxialForceBeam object (or null if there are no axial force beams in the model).

Return type

AxialForceBeam

Example

To get the first axial force beam in model m:

```
var afb = AxialForceBeam.First(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the axial force beams in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all axial force beams will be flagged in

- **flag** (Flag)

Flag to set on the axial force beams

Returns

No return value

Example

To flag all of the axial force beams with flag f in model m:

```
AxialForceBeam.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the axial force beam is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the axial force beam

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if axial force beam afb has flag f set on it:

```
if (afb.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each axial force beam in the model.
Note that ForEach has been designed to make looping over axial force beams as fast as possible and so has some limitations.
Firstly, a single temporary AxialForceBeam object is created and on each function call it is updated with the current axial force beam data. This means that you should not try to store the AxialForceBeam object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new axial force beams inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all axial force beams are in

- **func** (function)

Function to call for each axial force beam

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the axial force beams in model m:

```
AxialForceBeam.ForEach(m, test);
function test(afb)
{
// afb is AxialForceBeam object
}
```

To call function test for all of the axial force beams in model m with optional object:

```
var data = { x:0, y:0 };
AxialForceBeam.ForEach(m, test, data);
function test(afb, extra)
{
// afb is AxialForceBeam object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of AxialForceBeam objects or properties for all of the axial force beams in a model in PRIMER. If the optional property argument is not given then an array of AxialForceBeam objects is returned. If the property argument is given, that property value for each axial force beam is returned in the array instead of a AxialForceBeam object

Arguments

- **Model** (Model)

Model to get axial force beams from

- **property (optional)** (string)

Name for property to get for all axial force beams in the model

Returns

Array of AxialForceBeam objects or properties

Return type

Array

Example

To make an array of AxialForceBeam objects for all of the axial force beams in model m:

```
var a = AxialForceBeam.GetAll(m);
```

To return an array containing the value of property ’foo’ (for example ’x’ for a node) for each axial force beam in model m:

```
var a = AxialForceBeam.GetAll(m, ’foo’);
```

GetComments()

Description

Extracts the comments associated to a axial force beam.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the axial force beam afb:

```
var comm_array = afb.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of AxialForceBeam objects for all of the flagged axial force beams in a model in PRIMER If the optional property argument is not given then an array of AxialForceBeam objects is returned. If the property argument is given, then that property value for each axial force beam is returned in the array instead of a AxialForceBeam object

Arguments

- **Model** (Model)

Model to get axial force beams from

- **flag** (Flag)

Flag set on the axial force beams that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged axial force beams in the model

Returns

Array of AxialForceBeam objects or properties

Return type

Array

Example

To make an array of AxialForceBeam objects for all of the axial force beams in model m flagged with f:

```
var afb = AxialForceBeam.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the axial force beams in model m flagged with f:

```
var a = AxialForceBeam.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the AxialForceBeam object for a axial force beam ID.

Arguments

- **Model** (Model)

Model to find the axial force beam in

- **number** (integer)

number of the axial force beam you want the AxialForceBeam object for

Returns

AxialForceBeam object (or null if axial force beam does not exist).

Return type

AxialForceBeam

Example

To get the AxialForceBeam object for axial force beam 100 in model m

```
var afb = AxialForceBeam.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a AxialForceBeam property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the AxialForceBeam.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

axial force beam property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if AxialForceBeam property afb.example is a parameter:

```
Options.property_parameter_names = true;
if (afb.GetParameter(afb.example) ) do_something...
Options.property_parameter_names = false;
```

To check if AxialForceBeam property afb.example is a parameter by using the GetParameter method:

```
if (afb.ViewParameters().GetParameter(afb.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this axial force beam (*INITIAL_AXIAL_FORCE_BEAM). **Note that a carriage return is not added**. See also AxialForceBeam.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for axial force beam afb:

```
var key = afb.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the axial force beam. **Note that a carriage return is not added**. See also AxialForceBeam.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for axial force beam afb:

```
var cards = afb.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last axial force beam in the model.

Arguments

- **Model** (Model)

Model to get last axial force beam in

Returns

AxialForceBeam object (or null if there are no axial force beams in the model).

Return type

AxialForceBeam

Example

To get the last axial force beam in model m:

```
var afb = AxialForceBeam.Last(m);
```

Next()

Description

Returns the next axial force beam in the model.

Arguments

No arguments

Returns

AxialForceBeam object (or null if there are no more axial force beams in the model).

Return type

AxialForceBeam

Example

To get the axial force beam in model m after axial force beam afb:

```
var afb = afb.Next();
```

Previous()

Description

Returns the previous axial force beam in the model.

Arguments

No arguments

Returns

AxialForceBeam object (or null if there are no more axial force beams in the model).

Return type

AxialForceBeam

Example

To get the axial force beam in model m before axial force beam afb:

```
var afb = afb.Previous();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select axial force beams using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting axial force beams

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only axial force beams from that model can be selected. If the argument is a Flag then only axial force beams that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null,

any axial force beams can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of axial force beams selected or null if menu cancelled

Return type

Number

Example

To select axial force beams from model m, flagging those selected with flag f, giving the prompt 'Select axial force beams':

```
AxialForceBeam.Select(f, 'Select axial force beams', m);
```

To select axial force beams, flagging those selected with flag f but limiting selection to axial force beams flagged with flag l, giving the prompt 'Select axial force beams':

```
AxialForceBeam.Select(f, 'Select axial force beams', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the axial force beam.

Arguments

- **flag** (Flag)

Flag to set on the axial force beam

Returns

No return value

Example

To set flag f for axial force beam afb:

```
afb.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the axial force beam. The axial force beam will be sketched until you either call AxialForceBeam.Unsketch(), AxialForceBeam.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the axial force beam is sketched. If omitted redraw is true. If you want to sketch several axial force beams and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch axial force beam afb:

```
afb.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged axial force beams in the model. The axial force beams will be sketched until you either call AxialForceBeam.Unsketch(), AxialForceBeam.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged axial force beams will be sketched in

- **flag** (Flag)

Flag set on the axial force beams that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the axial force beams are sketched. If omitted redraw is true. If you want to sketch flagged axial force beams several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all axial force beams flagged with flag in model m:

```
AxialForceBeam.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of axial force beams in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing axial force beams should be counted. If false or omitted referenced but undefined axial force beams will also be included in the total.

Returns

number of axial force beams

Return type

Number

Example

To get the total number of axial force beams in model m:

```
var total = AxialForceBeam.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the axial force beams in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all axial force beams will be unset in

- **flag** (Flag)

Flag to unset on the axial force beams

Returns

No return value

Example

To unset the flag f on all the axial force beams in model m:

```
AxialForceBeam.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the axial force beam.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the axial force beam is unsketched. If omitted redraw is true. If you want to unsketch several axial force beams and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch axial force beam afb:

```
afb.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all axial force beams.

Arguments

- **Model** (Model)

Model that all axial force beams will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the axial force beams are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all axial force beams in model m:

```
AxialForceBeam.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged axial force beams in the model.

Arguments

- **Model** (Model)

Model that all axial force beams will be unsketched in

- **flag** (Flag)

Flag set on the axial force beams that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the axial force beams are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all axial force beams flagged with flag in model m:

```
AxialForceBeam.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with ’method chaining’ (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

AxialForceBeam object.

Return type

AxialForceBeam

Example

To check if AxialForceBeam property afb.example is a parameter by using the AxialForceBeam.GetParameter() method:

```
if (afb.ViewParameters().GetParameter(afb.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for axial force beam with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The axial force beam object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for axial force beam afb:

```
afb.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for axial force beam afb:

```
afb.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for axial force beam. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for axial force beam afb:

```
afb.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this axial force beam.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for axial force beam afb:

```
var xrefs = afb.Xrefs();
```

---

toString()

Description

Creates a string containing the axial force data in keyword format. Note that this contains the keyword header and the keyword cards. See also AxialForceBeam.Keyword() and AxialForceBeam.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for axial force beam afb in keyword format

```
var s = afb.toString();
```

StrainBeam class

The StrainBeam class gives you access to define initial strain beam cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- First(Model*[Model]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

StrainBeam properties

Name	Type	Description
eid	integer	Beam Element ID
exists (read only)	logical	true if initial strain beam exists, false if referred to but not defined.
include	integer	The Include file number that the initial strain beam is in.
model (read only)	integer	The Model number that the initial strain beam is in.

rdisp	real	The initial displacement along the local r-axis
rrot	real	The initial rotation about the local r-axis
sdisp	real	The initial displacement along the local s-axis
srot	real	The initial rotation about the local s-axis
tdisp	real	The initial displacement aling the local t-axis
trot	real	The initial rotation about the local t-axis

Detailed Description

The StrainBeam class allows you to create, modify, edit and manipulate strain_beam cards. See the documentation below for more details.

Constructor

new StrainBeam(Model*[Model]*, details*[object]*)

Description

Create a new StrainBeam object.

Arguments

- **Model** (Model)

Model that strain_beam will be created in

- **details** (object)

Details for creating the StrainBeam

Object has the following properties:

Name	Type	Description
eid	integer	Beam Element ID
rdisp (optional)	real	The initial displacement along the local r-axis
rrot (optional)	real	The initial rotation about the local r-axis
sdisp (optional)	real	The initial displacement along the local s-axis
srot (optional)	real	The initial rotation about the local s-axis
tdisp (optional)	real	The initial displacement along the local t-axis
trot (optional)	real	The initial rotation about the local t-axis

Returns

StrainBeam object

Return type

StrainBeam

Example

To create a new strain_beam in model m, using eid 1, rdisp 2.2, sdisp 3.3, tdisp 4.4, rrot 5.5, srot 6.6 and trot 7.7:

```
var s = new StrainBeam(m, {eid: 1, rdisp: 2.2, sdisp: 3.3, tdisp: 4.4, rrot: 5.5, srot: 6.6, trot: 7.7});
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a initial strain beam.

Arguments

- **Comment** (Comment)

Comment that will be attached to the initial strain beam

Returns

No return value

Example

To associate comment c to the initial strain beam ieb:

```
ieb.AssociateComment(c);
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the initial strain beam.

Arguments

- **flag** (Flag)

Flag to clear on the initial strain beam

Returns

No return value

Example

To clear flag f for initial strain beam ieb:

```
ieb.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the initial strain beam. The target include of the copied initial strain beam can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

StrainBeam object

Return type

StrainBeam

Example

To copy initial strain beam ieb into initial strain beam z:

```
var z = ieb.Copy();
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a initial strain beam.

Arguments

- **Comment** (Comment)

Comment that will be detached from the initial strain beam

Returns

No return value

Example

To detach comment c from the initial strain beam ieb:

```
ieb.DetachComment(c);
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for initial strain beam with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The initial strain beam object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for initial strain beam ieb:

```
ieb.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for initial strain beam ieb:

```
ieb.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for initial strain beam. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for initial strain beam ieb:

```
ieb.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first initial strain beam in the model.

Arguments

- **Model** (Model)

Model to get first initial strain beam in

Returns

StrainBeam object (or null if there are no initial strain beams in the model).

Return type

StrainBeam

Example

To get the first initial strain beam in model m:

```
var ieb = StrainBeam.First(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the initial strain beams in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all initial strain beams will be flagged in

- **flag** (Flag)

Flag to set on the initial strain beams

Returns

No return value

Example

To flag all of the initial strain beams with flag f in model m:

```
StrainBeam.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the initial strain beam is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the initial strain beam

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if initial strain beam ieb has flag f set on it:

```
if (ieb.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each initial strain beam in the model.
Note that ForEach has been designed to make looping over initial strain beams as fast as possible and so has some limitations.
Firstly, a single temporary StrainBeam object is created and on each function call it is updated with the current initial strain beam data. This means that you should not try to store the StrainBeam object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new initial strain beams inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all initial strain beams are in

- **func** (function)

Function to call for each initial strain beam

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the initial strain beams in model m:

```
StrainBeam.ForEach(m, test);
function test(ieb)
{
// ieb is StrainBeam object
}
```

To call function test for all of the initial strain beams in model m with optional object:

```
var data = { x:0, y:0 };
StrainBeam.ForEach(m, test, data);
function test(ieb, extra)
{
// ieb is StrainBeam object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of StrainBeam objects or properties for all of the initial strain beams in a model in PRIMER. If the optional property argument is not given then an array of StrainBeam objects is returned. If the property argument is given, that property value for each initial strain beam is returned in the array instead of a StrainBeam object

Arguments

- **Model** (Model)

Model to get initial strain beams from

- **property (optional)** (string)

Name for property to get for all initial strain beams in the model

Returns

Array of StrainBeam objects or properties

Return type

Array

Example

To make an array of StrainBeam objects for all of the initial strain beams in model m:

```
var a = StrainBeam.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each initial strain beam in model m:

```
var a = StrainBeam.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a initial strain beam.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the initial strain beam ieb:

```
var comm_array = ieb.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of StrainBeam objects for all of the flagged initial strain beams in a model in PRIMER If the optional property argument is not given then an array of StrainBeam objects is returned. If the property argument is given, then that property value for each initial strain beam is returned in the array instead of a StrainBeam object

Arguments

- **Model** (Model)

Model to get initial strain beams from

- **flag** (Flag)

Flag set on the initial strain beams that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged initial strain beams in the model

Returns

Array of StrainBeam objects or properties

Return type

Array

Example

To make an array of StrainBeam objects for all of the initial strain beams in model m flagged with f:

```
var ieb = StrainBeam.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the initial strain beams in model m flagged with f:

```
var a = StrainBeam.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the StrainBeam object for a initial strain beam ID.

Arguments

- **Model** (Model)

Model to find the initial strain beam in

- **number** (integer)

number of the initial strain beam you want the StrainBeam object for

Returns

StrainBeam object (or null if initial strain beam does not exist).

Return type

StrainBeam

Example

To get the StrainBeam object for initial strain beam 100 in model m

```
var ieb = StrainBeam.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a StrainBeam property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the StrainBeam.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

initial strain beam property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if StrainBeam property ieb.example is a parameter:

```
Options.property_parameter_names = true;
if (ieb.GetParameter(ieb.example) ) do_something...
Options.property_parameter_names = false;
```

To check if StrainBeam property ieb.example is a parameter by using the GetParameter method:

```
if (ieb.ViewParameters().GetParameter(ieb.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this initial strain beam (*INITIAL_STRAIN_SHELL). **Note that a carriage return is not added**. See also StrainBeam.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for strain_beam i:

```
var key = i.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the initial strain beam. **Note that a carriage return is not added**. See also StrainBeam.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for strain_beam i:

```
var cards = i.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last initial strain beam in the model.

Arguments

- **Model** (Model)

Model to get last initial strain beam in

Returns

StrainBeam object (or null if there are no initial strain beams in the model).

Return type

StrainBeam

Example

To get the last initial strain beam in model m:

```
var ieb = StrainBeam.Last(m);
```

Next()

Description

Returns the next initial strain beam in the model.

Arguments

No arguments

Returns

StrainBeam object (or null if there are no more initial strain beams in the model).

Return type

StrainBeam

Example

To get the initial strain beam in model m after initial strain beam ieb:

```
var ieb = ieb.Next();
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a initial strain beam.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only initial strain beams from that model can be picked. If the argument is a Flag then only initial strain beams that are flagged with *limit* can be selected. If omitted, or null, any initial strain beams from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

StrainBeam object (or null if not picked)

Return type

StrainBeam

Example

To pick a initial strain beam from model m giving the prompt 'Pick initial strain beam from screen':

```
var ieb = StrainBeam.Pick('Pick initial strain beam from screen', m);
```

Previous()

Description

Returns the previous initial strain beam in the model.

Arguments

No arguments

Returns

StrainBeam object (or null if there are no more initial strain beams in the model).

Return type

StrainBeam

Example

To get the initial strain beam in model m before initial strain beam ieb:

```
var ieb = ieb.Previous();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select initial strain beams using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting initial strain beams

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only initial strain beams from that model can be selected. If the argument is a Flag then only initial strain beams that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any initial strain beams can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of initial strain beams selected or null if menu cancelled

Return type

Number

Example

To select initial strain beams from model m, flagging those selected with flag f, giving the prompt 'Select initial strain beams':

```
StrainBeam.Select(f, 'Select initial strain beams', m);
```

To select initial strain beams, flagging those selected with flag f but limiting selection to initial strain beams flagged with flag l, giving the prompt 'Select initial strain beams':

```
StrainBeam.Select(f, 'Select initial strain beams', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the initial strain beam.

Arguments

- **flag** (Flag)

Flag to set on the initial strain beam

Returns

No return value

Example

To set flag f for initial strain beam ieb:

```
ieb.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the initial strain beam. The initial strain beam will be sketched until you either call StrainBeam.Unsketch(), StrainBeam.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the initial strain beam is sketched. If omitted redraw is true. If you want to sketch several initial strain beams and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch initial strain beam ieb:

```
ieb.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged initial strain beams in the model. The initial strain beams will be sketched until you either call StrainBeam.Unsketch(), StrainBeam.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged initial strain beams will be sketched in

- **flag** (Flag)

Flag set on the initial strain beams that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the initial strain beams are sketched. If omitted redraw is true. If you want to sketch flagged initial strain beams several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all initial strain beams flagged with flag in model m:

```
StrainBeam.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of initial strain beams in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing initial strain beams should be counted. If false or omitted referenced but undefined initial strain beams will also be included in the total.

Returns

number of initial strain beams

Return type

Number

Example

To get the total number of initial strain beams in model m:

```
var total = StrainBeam.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the initial strain beams in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all initial strain beams will be unset in

- **flag** (Flag)

Flag to unset on the initial strain beams

Returns

No return value

Example

To unset the flag f on all the initial strain beams in model m:

```
StrainBeam.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the initial strain beam.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the initial strain beam is unsketched. If omitted redraw is true. If you want to unsketch several initial strain beams and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch initial strain beam ieb:

```
ieb.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all initial strain beams.

Arguments

- **Model** (Model)

Model that all initial strain beams will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the initial strain beams are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all initial strain beams in model m:

```
StrainBeam.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged initial strain beams in the model.

Arguments

- **Model** (Model)

Model that all initial strain beams will be unsketched in

- **flag** (Flag)

Flag set on the initial strain beams that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the initial strain beams are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all initial strain beams flagged with flag in model m:

```
StrainBeam.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

StrainBeam object.

Return type

StrainBeam

Example

To check if StrainBeam property ieb.example is a parameter by using the StrainBeam.GetParameter() method:

```
if (ieb.ViewParameters().GetParameter(ieb.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for initial strain beam with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The initial strain beam object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for initial strain beam ieb:

```
ieb.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for initial strain beam ieb:

```
ieb.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for initial strain beam. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for initial strain beam ieb:

```
ieb.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this initial strain beam.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for initial strain beam ieb:

```
var xrefs = ieb.Xrefs();
```

---

toString()

Description

Creates a string containing the initial strain beam data in keyword format. Note that this contains the keyword header and the keyword cards. See also StrainBeam.Keyword() and StrainBeam.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for strain_beam i in keyword format

```
var s = i.toString();
```

StrainShell class

The StrainShell class gives you access to define initial strain shell cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- First(Model*[Model]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetIntegrationPoint(index*[integer]*)
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- SetIntegrationPoint(index*[integer]*, data*[Array of data]*)
- Sketch(redraw (optional)*[boolean]*)
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

StrainShell constants

Name	Description
StrainShell.SET	Initial is *INITIAL_STRAIN_SHELL_SET.
StrainShell.SHELL	Initial is *INITIAL_STRAIN_SHELL.

StrainShell properties

Name	Type	Description
eid	integer	Shell Element ID or shell set ID
exists (read only)	logical	true if initial strain shell exists, false if referred to but not defined.
ilocal	integer	Flag for coordinate system of strain components
include	integer	The Include file number that the initial strain shell is in.
large	logical	true if large format, false otherwise
model (read only)	integer	The Model number that the initial strain shell is in.
nplane	integer	Number of in plane integration points being output (not read when the SET option is used)
nthick	integer	Number of integration points through the thickness (not read when the SET option is used)
type	constant	The Intial strain shell type. Can be StrainShell.SHELL or StrainShell.SET.

Detailed Description

The StrainShell class allows you to create, modify, edit and manipulate strain_shell cards. See the documentation below for more details.

Constructor

new StrainShell(Model*[Model]*, type*[constant]*, eid*[integer]*, nplane*[integer]*, nthick*[integer]*, large*[integer]*, ilocal*[integer]*)

Description

Create a new StrainShell object.

Arguments

- **Model** (Model)

Model that strain_shell will be created in

- **type** (constant)

Specify the type of initial strain shell (Can be StrainShell.SHELL or StrainShell.SET)

- **eid** (integer)

Shell Element ID or shell set ID

- **nplane** (integer)

Number of in plane integration points being output

- **nthick** (integer)

Number of integration points through the thickness

- **large** (integer)

Large format flag, set 0 to turn it off or 1 to enable it. It is optional and set to 0 by default.

- **ilocal** (integer)

Flag for coordinate system of strain components. Set to 0 for global or 1 to enable local. It is optional and set to 0 by default

Returns

StrainShell object

Return type

StrainShell

Example

To create a new strain_shell in model m, of type SET

```
var s = new StrainShell(m, StrainShell.SET, 1, 2, 2);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a initial strain shell.

Arguments

- **Comment** (Comment)

Comment that will be attached to the initial strain shell

Returns

No return value

Example

To associate comment c to the initial strain shell iss:

```
iss.AssociateComment(c);
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the initial strain shell.

Arguments

- **flag** (Flag)

Flag to clear on the initial strain shell

Returns

No return value

Example

To clear flag f for initial strain shell iss:

```
iss.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the initial strain shell. The target include of the copied initial strain shell can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

StrainShell object

Return type

StrainShell

Example

To copy initial strain shell iss into initial strain shell z:

```
var z = iss.Copy();
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a initial strain shell.

Arguments

- **Comment** (Comment)

Comment that will be detached from the initial strain shell

Returns

No return value

Example

To detach comment c from the initial strain shell iss:

```
iss.DetachComment(c);
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for initial strain shell with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description

autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The initial strain shell object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for initial strain shell iss:

```
iss.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for initial strain shell iss:

```
iss.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for initial strain shell. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for initial strain shell iss:

```
iss.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first initial strain shell in the model.

Arguments

- **Model** (Model)

Model to get first initial strain shell in

Returns

StrainShell object (or null if there are no initial strain shells in the model).

Return type

StrainShell

Example

To get the first initial strain shell in model m:

```
var iss = StrainShell.First(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the initial strain shells in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all initial strain shells will be flagged in

- **flag** (Flag)

Flag to set on the initial strain shells

Returns

No return value

Example

To flag all of the initial strain shells with flag f in model m:

```
StrainShell.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the initial strain shell is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the initial strain shell

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if initial strain shell iss has flag f set on it:

```
if (iss.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each initial strain shell in the model.
Note that ForEach has been designed to make looping over initial strain shells as fast as possible and so has some limitations.
Firstly, a single temporary StrainShell object is created and on each function call it is updated with the current initial strain shell data. This means that you should not try to store the StrainShell object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new initial strain shells inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all initial strain shells are in

- **func** (function)

Function to call for each initial strain shell

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the initial strain shells in model m:

```
StrainShell.ForEach(m, test);
function test(iss)
{
// iss is StrainShell object
}
```

To call function test for all of the initial strain shells in model m with optional object:

```
var data = { x:0, y:0 };
StrainShell.ForEach(m, test, data);
function test(iss, extra)
{
// iss is StrainShell object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of StrainShell objects or properties for all of the initial strain shells in a model in PRIMER. If the optional property argument is not given then an array of StrainShell objects is returned. If the property argument is given, that property value for each initial strain shell is returned in the array instead of a StrainShell object

Arguments

- **Model** (Model)

Model to get initial strain shells from

- **property (optional)** (string)

Name for property to get for all initial strain shells in the model

Returns

Array of StrainShell objects or properties

Return type

Array

Example

To make an array of StrainShell objects for all of the initial strain shells in model m:

```
var a = StrainShell.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each initial strain shell in model m:

```
var a = StrainShell.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a initial strain shell.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the initial strain shell iss:

```
var comm_array = iss.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of StrainShell objects for all of the flagged initial strain shells in a model in PRIMER If the optional property argument is not given then an array of StrainShell objects is returned. If the property argument is given, then that property value for each initial strain shell is returned in the array instead of a StrainShell object

Arguments

- **Model** (Model)

Model to get initial strain shells from

- **flag** (Flag)

Flag set on the initial strain shells that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged initial strain shells in the model

Returns

Array of StrainShell objects or properties

Return type

Array

Example

To make an array of StrainShell objects for all of the initial strain shells in model m flagged with f:

```
var iss = StrainShell.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the initial strain shells in model m flagged with f:

```
var a = StrainShell.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the StrainShell object for a initial strain shell ID.

Arguments

- **Model** (Model)

Model to find the initial strain shell in

- **number** (integer)

number of the initial strain shell you want the StrainShell object for

Returns

StrainShell object (or null if initial strain shell does not exist).

Return type

StrainShell

Example

To get the StrainShell object for initial strain shell 100 in model m

```
var iss = StrainShell.GetFromID(m, 100);
```

GetIntegrationPoint(index*[integer]*)

Description

Returns the data for a specific integration point as an array. For each integration point there will be 7 strain component values. There are nplane x nthick integration points.

Arguments

- **index** (integer)

Index you want the integration point data for. **Note that indices start at 0**.

Returns

An array containing the integration point data.

Return type

Array

Example

To get the data for the 3rd integration point for initial strain shell iss:

```
var data = iss.GetIntegrationPoint(2);
```

GetParameter(prop*[string]*)

Description

Checks if a StrainShell property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the StrainShell.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

initial strain shell property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if StrainShell property iss.example is a parameter:

```
Options.property_parameter_names = true;
if (iss.GetParameter(iss.example) ) do_something...
Options.property_parameter_names = false;
```

To check if StrainShell property iss.example is a parameter by using the GetParameter method:

```
if (iss.ViewParameters().GetParameter(iss.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this initial strain shell (*INITIAL_STRAIN_SHELL). **Note that a carriage return is not added**. See also StrainShell.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for strain_shell i:

```
var key = i.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the initial strain shell. **Note that a carriage return is not added**. See also StrainShell.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for strain_shell i:

```
var cards = i.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last initial strain shell in the model.

Arguments

- **Model** (Model)

Model to get last initial strain shell in

Returns

StrainShell object (or null if there are no initial strain shells in the model).

Return type

StrainShell

Example

To get the last initial strain shell in model m:

```
var iss = StrainShell.Last(m);
```

Next()

Description

Returns the next initial strain shell in the model.

Arguments

No arguments

Returns

StrainShell object (or null if there are no more initial strain shells in the model).

Return type

StrainShell

Example

To get the initial strain shell in model m after initial strain shell iss:

```
var iss = iss.Next();
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a initial strain shell.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only initial strain shells from that model can be picked. If the argument is a Flag then only initial strain shells that are flagged with *limit* can be selected. If omitted, or null, any initial strain shells from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

StrainShell object (or null if not picked)

Return type

StrainShell

Example

To pick a initial strain shell from model m giving the prompt 'Pick initial strain shell from screen':

```
var iss = StrainShell.Pick('Pick initial strain shell from screen', m);
```

Previous()

Description

Returns the previous initial strain shell in the model.

Arguments

No arguments

Returns

StrainShell object (or null if there are no more initial strain shells in the model).

Return type

StrainShell

Example

To get the initial strain shell in model m before initial strain shell iss:

```
var iss = iss.Previous();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select initial strain shells using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting initial strain shells

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only initial strain shells from that model can be selected. If the argument is a Flag then only initial strain shells that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any initial strain shells can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of initial strain shells selected or null if menu cancelled

Return type

Number

Example

To select initial strain shells from model m, flagging those selected with flag f, giving the prompt 'Select initial strain shells':

```
StrainShell.Select(f, 'Select initial strain shells', m);
```

To select initial strain shells, flagging those selected with flag f but limiting selection to initial strain shells flagged with flag l, giving the prompt 'Select initial strain shells':

```
StrainShell.Select(f, 'Select initial strain shells', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the initial strain shell.

Arguments

- **flag** (Flag)

Flag to set on the initial strain shell

Returns

No return value

Example

To set flag f for initial strain shell iss:

```
iss.SetFlag(f);
```

SetIntegrationPoint(index*[integer]*, data*[Array of data]*)

Description

Set the data for a specific integration point. For each integration point there will be 7 strain component values. There are nplane x nthick integration points.

Arguments

- **index** (integer)

Index you want the integration point data for. **Note that indices start at 0**.

- **data** (Array of data)

Array containing the integration point data. The array length should be 7.

Returns

No return value.

Example

To set the 3rd integration point data for initial strain shell iss to the values in array adata:

```
iss.SetIntegrationPoint(2, adata);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the initial strain shell. The initial strain shell will be sketched until you either call StrainShell.Unsketch(), StrainShell.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the initial strain shell is sketched. If omitted redraw is true. If you want to sketch several initial strain shells and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch initial strain shell iss:

```
iss.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged initial strain shells in the model. The initial strain shells will be sketched until you either call StrainShell.Unsketch(), StrainShell.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged initial strain shells will be sketched in

- **flag** (Flag)

Flag set on the initial strain shells that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the initial strain shells are sketched. If omitted redraw is true. If you want to sketch flagged initial strain shells several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all initial strain shells flagged with flag in model m:

```
StrainShell.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of initial strain shells in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing initial strain shells should be counted. If false or omitted referenced but undefined initial strain shells will also be included in the total.

Returns

number of initial strain shells

Return type

Number

Example

To get the total number of initial strain shells in model m:

```
var total = StrainShell.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the initial strain shells in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all initial strain shells will be unset in

- **flag** (Flag)

Flag to unset on the initial strain shells

Returns

No return value

Example

To unset the flag f on all the initial strain shells in model m:

```
StrainShell.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the initial strain shell.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the initial strain shell is unsketched. If omitted redraw is true. If you want to unsketch several initial strain shells and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch initial strain shell iss:

```
iss.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all initial strain shells.

Arguments

- **Model** (Model)

Model that all initial strain shells will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the initial strain shells are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all initial strain shells in model m:

```
StrainShell.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged initial strain shells in the model.

Arguments

- **Model** (Model)

Model that all initial strain shells will be unsketched in

- **flag** (Flag)

Flag set on the initial strain shells that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the initial strain shells are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all initial strain shells flagged with flag in model m:

```
StrainShell.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

StrainShell object.

Return type

StrainShell

Example

To check if StrainShell property iss.example is a parameter by using the StrainShell.GetParameter() method:

```
if (iss.ViewParameters().GetParameter(iss.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for initial strain shell with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The initial strain shell object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for initial strain shell iss:

```
iss.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for initial strain shell iss:

```
iss.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for initial strain shell. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for initial strain shell iss:

```
iss.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this initial strain shell.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for initial strain shell iss:

```
var xrefs = iss.Xrefs();
```

---

toString()

Description

Creates a string containing the initial strain shell data in keyword format. Note that this contains the keyword header and the keyword cards. See also StrainShell.Keyword() and StrainShell.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for strain_shell i in keyword format

```
var s = i.toString();
```

StrainSolid class

The StrainSolid class gives you access to define initial strain solid cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- First(Model*[Model]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

StrainSolid constants

Name	Description
StrainSolid.SET	Initial is *INITIAL_STRESS_SOLID_SET.
StrainSolid.SOLID	Initial is *INITIAL_STRESS_SOLID.

StrainSolid properties

Name	Type	Description
eid	integer	Solid Element ID or solid set ID
epsxx	real	Define the xxth strain component in the global cartesian system.
epsxy	real	Define the xyth strain component in the global cartesian system.
epsyy	real	Define the yyth strain component in the global cartesian system.
epsyz	real	Define the yzth strain component in the global cartesian system.
epszx	real	Define the zxth strain component in the global cartesian system.
epszz	real	Define the zzth strain component in the global cartesian system.
exists (read only)	logical	true if initial strain solid exists, false if referred to but not defined.
include	integer	The Include file number that the initial strain solid is in.
model (read only)	integer	The Model number that the initial strain solid is in.
type	constant	The Intial strain solid type. Can be StrainSolid.SOLID or StrainSolid.SET.

Detailed Description

The StrainSolid class allows you to create, modify, edit and manipulate *INITIAL_STRESS_SOLID cards. See the documentation below for more details.

Constructor

new StrainSolid(Model*[Model]*, type*[constant]*, eid*[integer]*, epsxx*[real]*, epsyy*[real]*, epszz*[real]*, epsxy*[real]*, epsyz*[real]*, epszx*[real]*)

Description

Create a new StrainSolid object.

Arguments

- **Model** (Model)

Model that strain_solid will be created in

- **type** (constant)

Specify the type of initial strain solid (Can be StrainSolid.SOLID or StrainSolid.SET)

- **eid** (integer)

Solid Element ID or solid set ID

- **epsxx** (real)

The xxth strain component in the global cartesian system.

- **epsyy** (real)

The yyth strain component in the global cartesian system.

- **epszz** (real)

The zzth strain component in the global cartesian system.

- **epsxy** (real)

The xyth strain component in the global cartesian system.

- **epsyz** (real)

The yzth strain component in the global cartesian system.

- **epszx** (real)

The zxth strain component in the global cartesian system.

Returns

StrainSolid object

Return type

StrainSolid

Example

To create a new strain_solid in model m, of type SET with SOLID_SET id as 1, strain components as 10, 20, 30, 40, 50, 60.

```
var s = new StrainSolid(m, StrainSolid.SET, 1, 10, 20, 30, 40, 50, 60);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a initial strain solid.

Arguments

- **Comment** (Comment)

Comment that will be attached to the initial strain solid

Returns

No return value

Example

To associate comment c to the initial strain solid iso:

```
iso.AssociateComment(c);
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the initial strain solid.

Arguments

- **flag** (Flag)

Flag to clear on the initial strain solid

Returns

No return value

Example

To clear flag f for initial strain solid iso:

```
iso.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the initial strain solid. The target include of the copied initial strain solid can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

StrainSolid object

Return type

StrainSolid

Example

To copy initial strain solid iso into initial strain solid z:

```
var z = iso.Copy();
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a initial strain solid.

Arguments

- **Comment** (Comment)

Comment that will be detached from the initial strain solid

Returns

No return value

Example

To detach comment c from the initial strain solid iso:

```
iso.DetachComment(c);
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for initial strain solid with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The initial strain solid object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for initial strain solid iso:

```
iso.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for initial strain solid iso:

```
iso.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for initial strain solid. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for initial strain solid iso:

```
iso.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first initial strain solid in the model.

Arguments

- **Model** (Model)

Model to get first initial strain solid in

Returns

StrainSolid object (or null if there are no initial strain solids in the model).

Return type

StrainSolid

Example

To get the first initial strain solid in model m:

```
var iso = StrainSolid.First(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the initial strain solids in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all initial strain solids will be flagged in

- **flag** (Flag)

Flag to set on the initial strain solids

Returns

No return value

Example

To flag all of the initial strain solids with flag f in model m:

```
StrainSolid.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the initial strain solid is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the initial strain solid

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if initial strain solid iso has flag f set on it:

```
if (iso.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each initial strain solid in the model.
Note that ForEach has been designed to make looping over initial strain solids as fast as possible and so has some limitations.
Firstly, a single temporary StrainSolid object is created and on each function call it is updated with the current initial strain solid data. This means that you should not try to store the StrainSolid object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new initial strain solids inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all initial strain solids are in

- **func** (function)

Function to call for each initial strain solid

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the initial strain solids in model m:

```
StrainSolid.ForEach(m, test);
function test(iso)
{
// iso is StrainSolid object
}
```

To call function test for all of the initial strain solids in model m with optional object:

```
var data = { x:0, y:0 };
StrainSolid.ForEach(m, test, data);
function test(iso, extra)
{
// iso is StrainSolid object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of StrainSolid objects or properties for all of the initial strain solids in a model in PRIMER. If the optional property argument is not given then an array of StrainSolid objects is returned. If the property argument is given, that property value for each initial strain solid is returned in the array instead of a StrainSolid object

Arguments

- **Model** (Model)

Model to get initial strain solids from

- **property (optional)** (string)

Name for property to get for all initial strain solids in the model

Returns

Array of StrainSolid objects or properties

Return type

Array

Example

To make an array of StrainSolid objects for all of the initial strain solids in model m:

```
var a = StrainSolid.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each initial strain solid in model m:

```
var a = StrainSolid.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a initial strain solid.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the initial strain solid iso:

```
var comm_array = iso.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of StrainSolid objects for all of the flagged initial strain solids in a model in PRIMER If the optional property argument is not given then an array of StrainSolid objects is returned. If the property argument is given, then that property value for each initial strain solid is returned in the array instead of a StrainSolid object

Arguments

- **Model** (Model)

Model to get initial strain solids from

- **flag** (Flag)

Flag set on the initial strain solids that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged initial strain solids in the model

Returns

Array of StrainSolid objects or properties

Return type

Array

Example

To make an array of StrainSolid objects for all of the initial strain solids in model m flagged with f:

```
var iso = StrainSolid.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the initial strain solids in model m flagged with f:

```
var a = StrainSolid.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the StrainSolid object for a initial strain solid ID.

Arguments

- **Model** (Model)

Model to find the initial strain solid in

- **number** (integer)

number of the initial strain solid you want the StrainSolid object for

Returns

StrainSolid object (or null if initial strain solid does not exist).

Return type

StrainSolid

Example

To get the StrainSolid object for initial strain solid 100 in model m

```
var iso = StrainSolid.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a StrainSolid property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the StrainSolid.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

initial strain solid property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if StrainSolid property iso.example is a parameter:

```
Options.property_parameter_names = true;
if (iso.GetParameter(iso.example) ) do_something...
Options.property_parameter_names = false;
```

To check if StrainSolid property iso.example is a parameter by using the GetParameter method:

```
if (iso.ViewParameters().GetParameter(iso.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this initial strain solid (*INITIAL_STRESS_SOLID). **Note that a carriage return is not added**. See also StrainSolid.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for strain_solid i:

```
var key = i.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the initial strain solid. **Note that a carriage return is not added**. See also StrainSolid.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for strain_solid i:

```
var cards = i.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last initial strain solid in the model.

Arguments

- **Model** (Model)

Model to get last initial strain solid in

Returns

StrainSolid object (or null if there are no initial strain solids in the model).

Return type

StrainSolid

Example

To get the last initial strain solid in model m:

```
var iso = StrainSolid.Last(m);
```

Next()

Description

Returns the next initial strain solid in the model.

Arguments

No arguments

Returns

StrainSolid object (or null if there are no more initial strain solids in the model).

Return type

StrainSolid

Example

To get the initial strain solid in model m after initial strain solid iso:

```
var iso = iso.Next();
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a initial strain solid.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only initial strain solids from that model can be picked. If the argument is a Flag then only initial strain solids that are flagged with *limit* can be selected. If omitted, or null, any initial strain solids from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

StrainSolid object (or null if not picked)

Return type

StrainSolid

Example

To pick a initial strain solid from model m giving the prompt 'Pick initial strain solid from screen':

```
var iso = StrainSolid.Pick('Pick initial strain solid from screen', m);
```

Previous()

Description

Returns the previous initial strain solid in the model.

Arguments

No arguments

Returns

StrainSolid object (or null if there are no more initial strain solids in the model).

Return type

StrainSolid

Example

To get the initial strain solid in model m before initial strain solid iso:

```
var iso = iso.Previous();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select initial strain solids using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting initial strain solids

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only initial strain solids from that model can be selected. If the argument is a Flag then only initial strain solids that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any initial strain solids can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of initial strain solids selected or null if menu cancelled

Return type

Number

Example

To select initial strain solids from model m, flagging those selected with flag f, giving the prompt 'Select initial strain solids':

```
StrainSolid.Select(f, 'Select initial strain solids', m);
```

To select initial strain solids, flagging those selected with flag f but limiting selection to initial strain solids flagged with flag l, giving the prompt 'Select initial strain solids':

```
StrainSolid.Select(f, 'Select initial strain solids', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the initial strain solid.

Arguments

- **flag** (Flag)

Flag to set on the initial strain solid

Returns

No return value

Example

To set flag f for initial strain solid iso:

```
iso.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the initial strain solid. The initial strain solid will be sketched until you either call StrainSolid.Unsketch(), StrainSolid.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the initial strain solid is sketched. If omitted redraw is true. If you want to sketch several initial strain solids and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch initial strain solid iso:

```
iso.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged initial strain solids in the model. The initial strain solids will be sketched until you either call StrainSolid.Unsketch(), StrainSolid.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged initial strain solids will be sketched in

- **flag** (Flag)

Flag set on the initial strain solids that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the initial strain solids are sketched. If omitted redraw is true. If you want to sketch flagged initial strain solids several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all initial strain solids flagged with flag in model m:

```
StrainSolid.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of initial strain solids in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing initial strain solids should be counted. If false or omitted referenced but undefined initial strain solids will also be included in the total.

Returns

number of initial strain solids

Return type

Number

Example

To get the total number of initial strain solids in model m:

```
var total = StrainSolid.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the initial strain solids in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all initial strain solids will be unset in

- **flag** (Flag)

Flag to unset on the initial strain solids

Returns

No return value

Example

To unset the flag f on all the initial strain solids in model m:

```
StrainSolid.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the initial strain solid.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the initial strain solid is unsketched. If omitted redraw is true. If you want to unsketch several initial strain solids and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch initial strain solid iso:

```
iso.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all initial strain solids.

Arguments

- **Model** (Model)

Model that all initial strain solids will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the initial strain solids are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all initial strain solids in model m:

```
StrainSolid.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged initial strain solids in the model.

Arguments

- **Model** (Model)

Model that all initial strain solids will be unsketched in

- **flag** (Flag)

Flag set on the initial strain solids that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the initial strain solids are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all initial strain solids flagged with flag in model m:

```
StrainSolid.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

StrainSolid object.

Return type

StrainSolid

Example

To check if StrainSolid property iso.example is a parameter by using the StrainSolid.GetParameter() method:

```
if (iso.ViewParameters().GetParameter(iso.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for initial strain solid with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The initial strain solid object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for initial strain solid iso:

```
iso.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for initial strain solid iso:

```
iso.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for initial strain solid. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for initial strain solid iso:

```
iso.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this initial strain solid.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for initial strain solid iso:

```
var xrefs = iso.Xrefs();
```

---

toString()

Description

Creates a string containing the initial strain solid data in keyword format. Note that this contains the keyword header and the keyword cards. See also StrainSolid.Keyword() and StrainSolid.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for strain_solid i in keyword format

```
var s = i.toString();
```

StressBeam class

The StressBeam class gives you access to define initial stress beam cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- First(Model*[Model]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetIntegrationPoint(index*[integer]*)
- GetLocalAxesValues()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- SetIntegrationPoint(index*[integer]*, data*[Array of data]*)
- SetLocalAxesValues(data*[Array of data]*)
- Sketch(redraw (optional)*[boolean]*)
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

StressBeam constants

Constants for Coordinate system for stresses

Name	Description

StressBeam.GLOBAL_CSYS	Stress components are defined in the global coordinate system.
StressBeam.LOCAL_CSYS	Stress components are defined in the local beam system.

Constants for Number of Axes

Name	Description
StressBeam.NAXES_0	Number of variables giving beam local axes is zero.
StressBeam.NAXES_12	Number of variables giving beam local axes is 12.

Constants for Types of Rule

Name	Description
StressBeam.RULE_GUASS_QUADRATURE_NPTS_1	Type of Rule is 1 x 1 Gauss quadrature. Rule value is 1. Also sets npts to 1.
StressBeam.RULE_GUASS_QUADRATURE_NPTS_16	Type of Rule is 4 x 4 Gauss quadrature. Rule value is 5. Also sets npts to 16.
StressBeam.RULE_GUASS_QUADRATURE_NPTS_4	Type of Rule is 2 x 2 Gauss quadrature. Rule value is 2. Also sets npts to 4.
StressBeam.RULE_GUASS_QUADRATURE_NPTS_9	Type of Rule is 3 x 3 Gauss quadrature. Rule value is 3. Also sets npts to 9.
StressBeam.RULE_LOBATTO_QUADRATURE_NPTS_9	Type of Rule is 3 x 3 Lobatto quadrature. Rule value is 4. Also sets npts to 9.

StressBeam properties

Name	Type	Description
eid	integer	Beam Element ID
exists (read only)	logical	true if initial stress beam exists, false if referred to but not defined.
include	integer	The Include file number that the initial stress beam is in.
large	logical	true if large format, false otherwise.
local	constant	Coordinate system for stresses. Valid values are: StressBeam.GLOBAL_CSYS or StressBeam.LOCAL_CSYS.
model (read only)	integer	The Model number that the initial stress beam is in.
naxes	constant	Number of variables giving beam local axes. Valid values are: StressBeam.NAXES_0 or StressBeam.NAXES_12.
nhisv	integer	Number of additional history variables (only used if large is TRUE).
npts	integer	Number of integration points. The property value is set automatically if the rule is NOT set to a IntegrationBeam label.
rule	integer	Integration rule type number. Valid values are: StressBeam.RULE_GUASS_QUADRATURE_NPTS_1, StressBeam.RULE_GUASS_QUADRATURE_NPTS_4, StressBeam.RULE_GUASS_QUADRATURE_NPTS_9, StressBeam.RULE_LOBATTO_QUADRATURE_NPTS_9, StressBeam.RULE_GUASS_QUADRATURE_NPTS_16 or a IntegrationBeam label as a negative value.

Detailed Description

The StressBeam class allows you to create, modify, edit and manipulate *INITIAL_STRESS_BEAM cards. See the documentation below for more details.

Constructor

new StressBeam(Model*[Model]*, eid*[integer]*, rule*[integer]*, large (optional)*[boolean]*, nhisv (optional)*[integer]*, local (optional)*[constant]*, naxes (optional)*[constant]*)

Description

Create a new StressBeam object.

Arguments

- **Model** (Model)

Model that stress_beam will be created in

- **eid** (integer)

Beam Element ID

- **rule** (integer)

Integration rule type number. Valid values are: StressBeam.RULE_GUASS_QUADRATURE_NPTS_1, StressBeam.RULE_GUASS_QUADRATURE_NPTS_4, StressBeam.RULE_GUASS_QUADRATURE_NPTS_9, StressBeam.RULE_LOBATTO_QUADRATURE_NPTS_9, StressBeam.RULE_GUASS_QUADRATURE_NPTS_16 or a IntegrationBeam label as a negative value.

- **large (optional)** (boolean)

true if large format, false otherwise.

- **nhisv (optional)** (integer)

Number of additional history variables (only used if large is TRUE).

- **local (optional)** (constant)

Coordinate system for stresses. Valid values are: StressBeam.GLOBAL_CSYS or StressBeam.LOCAL_CSYS.

- **naxes (optional)** (constant)

Number of variables giving beam local axes. Valid values are: StressBeam.NAXES_0 or StressBeam.NAXES_12.

Returns

StressBeam object

Return type

StressBeam

Example

To create a new stress_beam in model m, for beam element id 100 with 4 number of integration points and number of history variables as 4:

```
var s = new StressBeam(m, 100, StressBeam.RULE_GUASS_QUADRATURE_NPTS_4, true,
4);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a initial stress beam.

Arguments

- **Comment** (Comment)

Comment that will be attached to the initial stress beam

Returns

No return value

Example

To associate comment c to the initial stress beam isb:

```
isb.AssociateComment(c);
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the initial stress beam.

Arguments

- **flag** (Flag)

Flag to clear on the initial stress beam

Returns

No return value

Example

To clear flag f for initial stress beam isb:

```
isb.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the initial stress beam. The target include of the copied initial stress beam can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

StressBeam object

Return type

StressBeam

Example

To copy initial stress beam isb into initial stress beam z:

```
var z = isb.Copy();
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a initial stress beam.

Arguments

- **Comment** (Comment)

Comment that will be detached from the initial stress beam

Returns

No return value

Example

To detach comment c from the initial stress beam isb:

```
isb.DetachComment(c);
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for initial stress beam with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The initial stress beam object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for initial stress beam isb:

```
isb.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for initial stress beam isb:

```
isb.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for initial stress beam. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for initial stress beam isb:

```
isb.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first initial stress beam in the model.

Arguments

- **Model** (Model)

Model to get first initial stress beam in

Returns

StressBeam object (or null if there are no initial stress beams in the model).

Return type

StressBeam

Example

To get the first initial stress beam in model m:

```
var isb = StressBeam.First(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the initial stress beams in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all initial stress beams will be flagged in

- **flag** (Flag)

Flag to set on the initial stress beams

Returns

No return value

Example

To flag all of the initial stress beams with flag f in model m:

```
StressBeam.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the initial stress beam is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the initial stress beam

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if initial stress beam isb has flag f set on it:

```
if (isb.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each initial stress beam in the model.
Note that ForEach has been designed to make looping over initial stress beams as fast as possible and so has some limitations.
Firstly, a single temporary StressBeam object is created and on each function call it is updated with the current initial stress beam data. This means that you should not try to store the StressBeam object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new initial stress beams inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all initial stress beams are in

- **func** (function)

Function to call for each initial stress beam

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the initial stress beams in model m:

```
StressBeam.ForEach(m, test);
function test(isb)
{
// isb is StressBeam object
}
```

To call function test for all of the initial stress beams in model m with optional object:

```
var data = { x:0, y:0 };
StressBeam.ForEach(m, test, data);
function test(isb, extra)
{
// isb is StressBeam object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of StressBeam objects or properties for all of the initial stress beams in a model in PRIMER. If the optional property argument is not given then an array of StressBeam objects is returned. If the property argument is given, that property value for each initial stress beam is returned in the array instead of a StressBeam object

Arguments

- **Model** (Model)

Model to get initial stress beams from

- **property (optional)** (string)

Name for property to get for all initial stress beams in the model

Returns

Array of StressBeam objects or properties

Return type

Array

Example

To make an array of StressBeam objects for all of the initial stress beams in model m:

```
var a = StressBeam.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each initial stress beam in model m:

```
var a = StressBeam.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a initial stress beam.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the initial stress beam isb:

```
var comm_array = isb.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of StressBeam objects for all of the flagged initial stress beams in a model in PRIMER If the optional property argument is not given then an array of StressBeam objects is returned. If the property argument is given, then that property value for each initial stress beam is returned in the array instead of a StressBeam object

Arguments

- **Model** (Model)

Model to get initial stress beams from

- **flag** (Flag)

Flag set on the initial stress beams that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged initial stress beams in the model

Returns

Array of StressBeam objects or properties

Return type

Array

Example

To make an array of StressBeam objects for all of the initial stress beams in model m flagged with f:

```
var isb = StressBeam.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the initial stress beams in model m flagged with f:

```
var a = StressBeam.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the StressBeam object for a initial stress beam ID.

Arguments

- **Model** (Model)

Model to find the initial stress beam in

- **number** (integer)

number of the initial stress beam you want the StressBeam object for

Returns

StressBeam object (or null if initial stress beam does not exist).

Return type

StressBeam

Example

To get the StressBeam object for initial stress beam 100 in model m

```
var isb = StressBeam.GetFromID(m, 100);
```

GetIntegrationPoint(index*[integer]*)

Description

Returns the data for a specific integration point as an array. For each integration point there will be 7 values if large is FALSE. For each integration point there will be (7 + nhisv) values if large is TRUE. There are npts integration points.

Arguments

- **index** (integer)

Index you want the integration point data for. **Note that indices start at 0**.

Returns

An array containing the integration point data.

Return type

Array

Example

To get the data for the 3rd integration point for initial stress beam isb:

```
var data = isb.GetIntegrationPoint(2);
```

GetLocalAxesValues()

Description

Returns the 12 axes values as an array. The axes values are valid only if the naxes is set to StressBeam.NAXES_12.

Arguments

No arguments

Returns

An array containing the axes values.

Return type

Array

Example

To get the data for the axes values for initial stress beam isb:

```
var data = isb.GetLocalAxesValues();
```

GetParameter(prop*[string]*)

Description

Checks if a StressBeam property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the StressBeam.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

initial stress beam property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if StressBeam property isb.example is a parameter:

```
Options.property_parameter_names = true;
if (isb.GetParameter(isb.example) ) do_something...
Options.property_parameter_names = false;
```

To check if StressBeam property isb.example is a parameter by using the GetParameter method:

```
if (isb.ViewParameters().GetParameter(isb.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this initial stress beam (*INITIAL_STRESS_BEAM). **Note that a carriage return is not added**. See also StressBeam.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for stress_beam i:

```
var key = i.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the initial stress beam. **Note that a carriage return is not added**. See also StressBeam.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for stress_beam i:

```
var cards = i.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last initial stress beam in the model.

Arguments

- **Model** (Model)

Model to get last initial stress beam in

Returns

StressBeam object (or null if there are no initial stress beams in the model).

Return type

StressBeam

Example

To get the last initial stress beam in model m:

```
var isb = StressBeam.Last(m);
```

Next()

Description

Returns the next initial stress beam in the model.

Arguments

No arguments

Returns

StressBeam object (or null if there are no more initial stress beams in the model).

Return type

StressBeam

Example

To get the initial stress beam in model m after initial stress beam isb:

```
var isb = isb.Next();
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a initial stress beam.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only initial stress beams from that model can be picked. If the argument is a Flag then only initial stress beams that are flagged with *limit* can be selected. If omitted, or null, any initial stress beams from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

StressBeam object (or null if not picked)

Return type

StressBeam

Example

To pick a initial stress beam from model m giving the prompt 'Pick initial stress beam from screen':

```
var isb = StressBeam.Pick('Pick initial stress beam from screen', m);
```

Previous()

Description

Returns the previous initial stress beam in the model.

Arguments

No arguments

Returns

StressBeam object (or null if there are no more initial stress beams in the model).

Return type

StressBeam

Example

To get the initial stress beam in model m before initial stress beam isb:

```
var isb = isb.Previous();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select initial stress beams using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting initial stress beams

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only initial stress beams from that model can be selected. If the argument is a Flag then only initial stress beams that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any initial stress beams can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of initial stress beams selected or null if menu cancelled

Return type

Number

Example

To select initial stress beams from model m, flagging those selected with flag f, giving the prompt 'Select initial stress beams':

```
StressBeam.Select(f, 'Select initial stress beams', m);
```

To select initial stress beams, flagging those selected with flag f but limiting selection to initial stress beams flagged with flag l, giving the prompt 'Select initial stress beams':

```
StressBeam.Select(f, 'Select initial stress beams', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the initial stress beam.

Arguments

- **flag** (Flag)

Flag to set on the initial stress beam

Returns

No return value

Example

To set flag f for initial stress beam isb:

```
isb.SetFlag(f);
```

SetIntegrationPoint(index*[integer]*, data*[Array of data]*)

Description

Set the data for a specific integration point. For each integration point there will be 7 values if large is FALSE. For each integration point there will be (7 + nhisv) values if large is TRUE. There are npts integration points.

Arguments

- **index** (integer)

Index you want the integration point data for. **Note that indices start at 0**.

- **data** (Array of data)

Array containing the integration point data. The array length should be 7 if large is FALSE. The array length should be (7 + nhisv) if large is TRUE.

Returns

No return value.

Example

To set the 3rd integration point data for initial stress beam isb to the values in array adata:

```
isb.SetIntegrationPoint(2, adata);
```

SetLocalAxesValues(data*[Array of data]*)

Description

Sets the 12 axes values as an array. The axes values are set only if the naxes is set to StressBeam.NAXES_12.

Arguments

- **data** (Array of data)

Array containing the axes values data. The array length should be 12.

Returns

No return value

Example

To set the data for the axes values for initial stress beam isb:

```
var data = isb.SetLocalAxesValues(data);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the initial stress beam. The initial stress beam will be sketched until you either call StressBeam.Unsketch(), StressBeam.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the initial stress beam is sketched. If omitted redraw is true. If you want to sketch several initial stress beams and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch initial stress beam isb:

```
isb.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged initial stress beams in the model. The initial stress beams will be sketched until you either call StressBeam.Unsketch(), StressBeam.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged initial stress beams will be sketched in

- **flag** (Flag)

Flag set on the initial stress beams that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the initial stress beams are sketched. If omitted redraw is true. If you want to sketch flagged initial stress beams several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all initial stress beams flagged with flag in model m:

```
StressBeam.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of initial stress beams in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing initial stress beams should be counted. If false or omitted referenced but undefined initial stress beams will also be included in the total.

Returns

number of initial stress beams

Return type

Number

Example

To get the total number of initial stress beams in model m:

```
var total = StressBeam.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the initial stress beams in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all initial stress beams will be unset in

- **flag** (Flag)

Flag to unset on the initial stress beams

Returns

No return value

Example

To unset the flag f on all the initial stress beams in model m:

```
StressBeam.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the initial stress beam.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the initial stress beam is unsketched. If omitted redraw is true. If you want to unsketch several initial stress beams and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch initial stress beam isb:

```
isb.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all initial stress beams.

Arguments

- **Model** (Model)

Model that all initial stress beams will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the initial stress beams are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all initial stress beams in model m:

```
StressBeam.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged initial stress beams in the model.

Arguments

- **Model** (Model)

Model that all initial stress beams will be unsketched in

- **flag** (Flag)

Flag set on the initial stress beams that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the initial stress beams are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all initial stress beams flagged with flag in model m:

```
StressBeam.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

StressBeam object.

Return type

StressBeam

Example

To check if StressBeam property isb.example is a parameter by using the StressBeam.GetParameter() method:

```
if (isb.ViewParameters().GetParameter(isb.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for initial stress beam with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The initial stress beam object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...

autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for initial stress beam isb:

```
isb.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for initial stress beam isb:

```
isb.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for initial stress beam. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for initial stress beam isb:

```
isb.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this initial stress beam.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for initial stress beam isb:

```
var xrefs = isb.Xrefs();
```

---

toString()

Description

Creates a string containing the initial stress beam data in keyword format. Note that this contains the keyword header and the keyword cards. See also StressBeam.Keyword() and StressBeam.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for stress_beam i in keyword format

```
var s = i.toString();
```

StressSection class

The StressSection class gives you access to define *INITIAL_STRESS_SECTION cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()

- toString()

StressSection properties

Name	Type	Description
csid	integer	Cross section ID.
exists (read only)	logical	true if stress section exists, false if referred to but not defined.
include	integer	The Include file number that the stress section is in.
issid	integer	StressSection number. Also see the label property which is an alternative name for this.
istiff	integer	Load curve ID defining the artificial stress fraction versus time.
izshear	integer	Shear stress flag.
label	integer	StressSection number. Also see the issid property which is an alternative name for this.
lcid	integer	Load curve ID defining preload stress versus time.
model (read only)	integer	The Model number that the stress section is in.
psid	integer	Part set ID.
vid	integer	Vector ID defining the direction normal to the cross section.

Detailed Description

The StressSection class allows you to create, modify, edit and manipulate initial stress section cards. See the documentation below for more details.

Constructor

new StressSection(Model*[Model]*, issid*[integer]*, csid*[integer]*, lcid*[integer]*, psid*[integer]*, vid*[integer]*, izshear*[integer]*, istiff (optional)*[integer]*)

Description

Create a new StressSection object.

Arguments

- **Model** (Model)

Model that stress section will be created in

- **issid** (integer)

StressSection number.

- **csid** (integer)

Cross section ID.

- **lcid** (integer)

Load curve ID defining preload stress versus time.

- **psid** (integer)

Part set ID.

- **vid** (integer)

Vector ID.

- **izshear** (integer)

Shear stress flag.

- **istiff (optional)** (integer)

Load curve ID defining artificial stress fraction versus time.

Returns

StressSection object

Return type

StressSection

Example

To create a new stress section in model m with label 11, cross section 12, load curve 13, part set 14, vector 15 and shear stress flag 16:

```
var iss = new StressSection(m, 11, 12, 13, 14, 15, 16);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a stress section.

Arguments

- **Comment** (Comment)

Comment that will be attached to the stress section

Returns

No return value

Example

To associate comment c to the stress section iss:

```
iss.AssociateComment(c);
```

Blank()

Description

Blanks the stress section

Arguments

No arguments

Returns

No return value

Example

To blank stress section iss:

```
iss.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the stress sections in the model.

Arguments

- **Model** (Model)

Model that all stress sections will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the stress sections in model m:

```
StressSection.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged stress sections in the model.

Arguments

- **Model** (Model)

Model that all the flagged stress sections will be blanked in

- **flag** (Flag)

Flag set on the stress sections that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the stress sections in model m flagged with f:

```
StressSection.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the stress section is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if stress section iss is blanked:

```
if (iss.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse stress section iss:

```
iss.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the stress section.

Arguments

- **flag** (Flag)

Flag to clear on the stress section

Returns

No return value

Example

To clear flag f for stress section iss:

```
iss.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the stress section. The target include of the copied stress section can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

StressSection object

Return type

StressSection

Example

To copy stress section iss into stress section z:

```
var z = iss.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a stress section

Arguments

- **Model** (Model)

Model that the stress section will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

StressSection object (or null if not made).

Return type

StressSection

Example

To start creating a stress section in model m:

```
var iss = StressSection.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a stress section.

Arguments

- **Comment** (Comment)

Comment that will be detached from the stress section

Returns

No return value

Example

To detach comment c from the stress section iss:

```
iss.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit stress section iss:

```
iss.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for stress section with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The stress section object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for stress section iss:

```
iss.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for stress section iss:

```
iss.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for stress section. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for stress section iss:

```
iss.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first stress section in the model.

Arguments

- **Model** (Model)

Model to get first stress section in

Returns

StressSection object (or null if there are no stress sections in the model).

Return type

StressSection

Example

To get the first stress section in model m:

```
var iss = StressSection.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free stress section label in the model. Also see StressSection.LastFreeLabel(), StressSection.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free stress section label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

StressSection label.

Return type

Number

Example

To get the first free stress section label in model m:

```
var label = StressSection.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the stress sections in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all stress sections will be flagged in

- **flag** (Flag)

Flag to set on the stress sections

Returns

No return value

Example

To flag all of the stress sections with flag f in model m:

```
StressSection.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the stress section is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the stress section

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if stress section iss has flag f set on it:

```
if (iss.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each stress section in the model.
Note that ForEach has been designed to make looping over stress sections as fast as possible and so has some limitations.
Firstly, a single temporary StressSection object is created and on each function call it is updated with the current stress section data. This means that you should not try to store the StressSection object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new stress sections inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all stress sections are in

- **func** (function)

Function to call for each stress section

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the stress sections in model m:

```
StressSection.ForEach(m, test);
function test(iss)
{
// iss is StressSection object
}
```

To call function test for all of the stress sections in model m with optional object:

```
var data = { x:0, y:0 };
StressSection.ForEach(m, test, data);
function test(iss, extra)
{
// iss is StressSection object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of StressSection objects or properties for all of the stress sections in a model in PRIMER. If the optional property argument is not given then an array of StressSection objects is returned. If the property argument is given, that property value for each stress section is returned in the array instead of a StressSection object

Arguments

- **Model** (Model)

Model to get stress sections from

- **property (optional)** (string)

Name for property to get for all stress sections in the model

Returns

Array of StressSection objects or properties

Return type

Array

Example

To make an array of StressSection objects for all of the stress sections in model m:

```
var a = StressSection.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each stress section in model m:

```
var a = StressSection.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a stress section.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the stress section iss:

```
var comm_array = iss.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of StressSection objects for all of the flagged stress sections in a model in PRIMER If the optional property argument is not given then an array of StressSection objects is returned. If the property argument is given, then that property value for each stress section is returned in the array instead of a StressSection object

Arguments

- **Model** (Model)

Model to get stress sections from

- **flag** (Flag)

Flag set on the stress sections that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged stress sections in the model

Returns

Array of StressSection objects or properties

Return type

Array

Example

To make an array of StressSection objects for all of the stress sections in model m flagged with f:

```
var iss = StressSection.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the stress sections in model m flagged with f:

```
var a = StressSection.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the StressSection object for a stress section ID.

Arguments

- **Model** (Model)

Model to find the stress section in

- **number** (integer)

number of the stress section you want the StressSection object for

Returns

StressSection object (or null if stress section does not exist).

Return type

StressSection

Example

To get the StressSection object for stress section 100 in model m

```
var iss = StressSection.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a StressSection property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the StressSection.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

stress section property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if StressSection property iss.example is a parameter:

```
Options.property_parameter_names = true;
if (iss.GetParameter(iss.example) ) do_something...
Options.property_parameter_names = false;
```

To check if StressSection property iss.example is a parameter by using the GetParameter method:

```
if (iss.ViewParameters().GetParameter(iss.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this stress section. **Note that a carriage return is not added**. See also StressSection.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for stress section iss:

```
var key = iss.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the stress section. **Note that a carriage return is not added**. See also StressSection.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for stress section iss:

```
var cards = iss.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last stress section in the model.

Arguments

- **Model** (Model)

Model to get last stress section in

Returns

StressSection object (or null if there are no stress sections in the model).

Return type

StressSection

Example

To get the last stress section in model m:

```
var iss = StressSection.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free stress section label in the model. Also see StressSection.FirstFreeLabel(), StressSection.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free stress section label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

StressSection label.

Return type

Number

Example

To get the last free stress section label in model m:

```
var label = StressSection.LastFreeLabel(m);
```

Next()

Description

Returns the next stress section in the model.

Arguments

No arguments

Returns

StressSection object (or null if there are no more stress sections in the model).

Return type

StressSection

Example

To get the stress section in model m after stress section iss:

```
var iss = iss.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) stress section label in the model. Also see StressSection.FirstFreeLabel(), StressSection.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free stress section label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

StressSection label.

Return type

Number

Example

To get the next free stress section label in model m:

```
var label = StressSection.NextFreeLabel(m);
```

Previous()

Description

Returns the previous stress section in the model.

Arguments

No arguments

Returns

StressSection object (or null if there are no more stress sections in the model).

Return type

StressSection

Example

To get the stress section in model m before stress section iss:

```
var iss = iss.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the stress sections in the model.

Arguments

- **Model** (Model)

Model that all stress sections will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the stress sections in model m, from 1000000:

```
StressSection.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged stress sections in the model.

Arguments

- **Model** (Model)

Model that all the flagged stress sections will be renumbered in

- **flag** (Flag)

Flag set on the stress sections that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the stress sections in model m flagged with f, from 1000000:

```
StressSection.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select stress sections using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting stress sections

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only stress sections from that model can be selected. If the argument is a Flag then only stress sections that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any stress sections can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of stress sections selected or null if menu cancelled

Return type

Number

Example

To select stress sections from model m, flagging those selected with flag f, giving the prompt 'Select stress sections':

```
StressSection.Select(f, 'Select stress sections', m);
```

To select stress sections, flagging those selected with flag f but limiting selection to stress sections flagged with flag l, giving the prompt 'Select stress sections':

```
StressSection.Select(f, 'Select stress sections', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the stress section.

Arguments

- **flag** (Flag)

Flag to set on the stress section

Returns

No return value

Example

To set flag f for stress section iss:

```
iss.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the stress section. The stress section will be sketched until you either call StressSection.Unsketch(), StressSection.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the stress section is sketched. If omitted redraw is true. If you want to sketch several stress sections and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch stress section iss:

```
iss.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged stress sections in the model. The stress sections will be sketched until you either call StressSection.Unsketch(), StressSection.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged stress sections will be sketched in

- **flag** (Flag)

Flag set on the stress sections that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the stress sections are sketched. If omitted redraw is true. If you want to sketch flagged stress sections several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all stress sections flagged with flag in model m:

```
StressSection.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of stress sections in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing stress sections should be counted. If false or omitted referenced but undefined stress sections will also be included in the total.

Returns

number of stress sections

Return type

Number

Example

To get the total number of stress sections in model m:

```
var total = StressSection.Total(m);
```

Unblank()

Description

Unblanks the stress section

Arguments

No arguments

Returns

No return value

Example

To unblank stress section iss:

```
iss.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the stress sections in the model.

Arguments

- **Model** (Model)

Model that all stress sections will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the stress sections in model m:

```
StressSection.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged stress sections in the model.

Arguments

- **Model** (Model)

Model that the flagged stress sections will be unblanked in

- **flag** (Flag)

Flag set on the stress sections that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the stress sections in model m flagged with f:

```
StressSection.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the stress sections in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all stress sections will be unset in

- **flag** (Flag)

Flag to unset on the stress sections

Returns

No return value

Example

To unset the flag f on all the stress sections in model m:

```
StressSection.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the stress section.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the stress section is unsketched. If omitted redraw is true. If you want to unsketch several stress sections and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch stress section iss:

```
iss.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all stress sections.

Arguments

- **Model** (Model)

Model that all stress sections will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the stress sections are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all stress sections in model m:

```
StressSection.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged stress sections in the model.

Arguments

- **Model** (Model)

Model that all stress sections will be unsketched in

- **flag** (Flag)

Flag set on the stress sections that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the stress sections are unsketched. If omitted redraw is true. If you want to

unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all stress sections flagged with flag in model m:

```
StressSection.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

StressSection object.

Return type

StressSection

Example

To check if StressSection property iss.example is a parameter by using the StressSection.GetParameter() method:

```
if (iss.ViewParameters().GetParameter(iss.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for stress section with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The stress section object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...

autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for stress section iss:

```
iss.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for stress section iss:

```
iss.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for stress section. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for stress section iss:

```
iss.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this stress section.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for stress section iss:

```
var xrefs = iss.Xrefs();
```

---

toString()

Description

Creates a string containing the stress section data in keyword format. Note that this contains the keyword header and the keyword cards. See also StressSection.Keyword() and StressSection.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for stress section iss in keyword format

```
var s = iss.toString();
```

StressShell class

The StressShell class gives you access to define initial stress shell cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- First(Model*[Model]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetIntegrationPoint(index*[integer]*)
- GetParameter(prop*[string]*)
- GetThermalIntegrationPoint(index*[integer]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- SetIntegrationPoint(index*[integer]*, data*[Array of data]*)
- SetThermalIntegrationPoint(index*[integer]*, data*[Array of data]*)
- Sketch(redraw (optional)*[boolean]*)
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

StressShell constants

Name	Description
StressShell.SET	Initial is *INITIAL_STRESS_SHELL_SET.
StressShell.SHELL	Initial is *INITIAL_STRESS_SHELL.

StressShell properties

Name	Type	Description
eid	integer	Shell Element ID or shell set ID
exists (read only)	logical	true if initial stress shell exists, false if referred to but not defined.
include	integer	The Include file number that the initial stress shell is in.
large	logical	true if large format, false otherwise
model (read only)	integer	The Model number that the initial stress shell is in.
nhisv	integer	Number of additional history variables
nplane	integer	Number of in plane integration points being output
ntensr	integer	Number of components of tensor data taken from the element history variables stored
nthhsv	integer	Number of thermal history variables per thermal integration point
nthick	integer	Number of integration points through the thickness
nthint	integer	Number of thermal integration points
type	constant	The Intial stress shell type. Can be StressShell.SHELL or StressShell.SET.

Detailed Description

The StressShell class allows you to create, modify, edit and manipulate stress_shell cards. See the documentation below for more details.

Constructor

new StressShell(Model*[Model]*, type*[constant]*, eid*[integer]*, nplane*[integer]*, nthick*[integer]*, nhisv*[integer]*, ntensr*[integer]*)

Description

Create a new StressShell object.

Arguments

- **Model** (Model)

Model that stress_shell will be created in

- **type** (constant)

Specify the type of initial stress shell (Can be StressShell.SHELL or StressShell.SET)

- **eid** (integer)

Shell Element ID or shell set ID

- **nplane** (integer)

Number of in plane integration points being output

- **nthick** (integer)

Number of integration points through the thickness

- **nhisv** (integer)

Number of additional history variables

- **ntensr** (integer)

Number of components of tensor data taken from the element history variables stored

Returns

StressShell object

Return type

StressShell

Example

To create a new stress_shell in model m, of type SET

```
var s = new StressShell(m, StressShell.SET, 1, 3, 0, 0);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a initial stress shell.

Arguments

- **Comment** (Comment)

Comment that will be attached to the initial stress shell

Returns

No return value

Example

To associate comment c to the initial stress shell iss:

```
iss.AssociateComment(c);
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the initial stress shell.

Arguments

- **flag** (Flag)

Flag to clear on the initial stress shell

Returns

No return value

Example

To clear flag f for initial stress shell iss:

```
iss.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the initial stress shell. The target include of the copied initial stress shell can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

StressShell object

Return type

StressShell

Example

To copy initial stress shell iss into initial stress shell z:

```
var z = iss.Copy();
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a initial stress shell.

Arguments

- **Comment** (Comment)

Comment that will be detached from the initial stress shell

Returns

No return value

Example

To detach comment c from the initial stress shell iss:

```
iss.DetachComment(c);
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for initial stress shell with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description

autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The initial stress shell object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for initial stress shell iss:

```
iss.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for initial stress shell iss:

```
iss.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for initial stress shell. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for initial stress shell iss:

```
iss.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first initial stress shell in the model.

Arguments

- **Model** (Model)

Model to get first initial stress shell in

Returns

StressShell object (or null if there are no initial stress shells in the model).

Return type

StressShell

Example

To get the first initial stress shell in model m:

```
var iss = StressShell.First(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the initial stress shells in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all initial stress shells will be flagged in

- **flag** (Flag)

Flag to set on the initial stress shells

Returns

No return value

Example

To flag all of the initial stress shells with flag f in model m:

```
StressShell.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the initial stress shell is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the initial stress shell

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if initial stress shell iss has flag f set on it:

```
if (iss.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each initial stress shell in the model.
Note that ForEach has been designed to make looping over initial stress shells as fast as possible and so has some limitations.
Firstly, a single temporary StressShell object is created and on each function call it is updated with the current initial stress shell data. This means that you should not try to store the StressShell object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new initial stress shells inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all initial stress shells are in

- **func** (function)

Function to call for each initial stress shell

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the initial stress shells in model m:

```
StressShell.ForEach(m, test);
function test(iss)
{
// iss is StressShell object
}
```

To call function test for all of the initial stress shells in model m with optional object:

```
var data = { x:0, y:0 };
StressShell.ForEach(m, test, data);
function test(iss, extra)
{
// iss is StressShell object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of StressShell objects or properties for all of the initial stress shells in a model in PRIMER. If the optional property argument is not given then an array of StressShell objects is returned. If the property argument is given, that property value for each initial stress shell is returned in the array instead of a StressShell object

Arguments

- **Model** (Model)

Model to get initial stress shells from

- **property (optional)** (string)

Name for property to get for all initial stress shells in the model

Returns

Array of StressShell objects or properties

Return type

Array

Example

To make an array of StressShell objects for all of the initial stress shells in model m:

```
var a = StressShell.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each initial stress shell in model m:

```
var a = StressShell.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a initial stress shell.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the initial stress shell iss:

```
var comm_array = iss.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of StressShell objects for all of the flagged initial stress shells in a model in PRIMER If the optional property argument is not given then an array of StressShell objects is returned. If the property argument is given, then that property value for each initial stress shell is returned in the array instead of a StressShell object

Arguments

- **Model** (Model)

Model to get initial stress shells from

- **flag** (Flag)

Flag set on the initial stress shells that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged initial stress shells in the model

Returns

Array of StressShell objects or properties

Return type

Array

Example

To make an array of StressShell objects for all of the initial stress shells in model m flagged with f:

```
var iss = StressShell.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the initial stress shells in model m flagged with f:

```
var a = StressShell.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the StressShell object for a initial stress shell ID.

Arguments

- **Model** (Model)

Model to find the initial stress shell in

- **number** (integer)

number of the initial stress shell you want the StressShell object for

Returns

StressShell object (or null if initial stress shell does not exist).

Return type

StressShell

Example

To get the StressShell object for initial stress shell 100 in model m

```
var iss = StressShell.GetFromID(m, 100);
```

GetIntegrationPoint(index*[integer]*)

Description

Returns the data for a specific integration point as an array. For each integration point there will be 8 + nhisv + (6 x ntensr) values. There are nplane x nthick integration points.

Arguments

- **index** (integer)

Index you want the integration point data for. **Note that indices start at 0**.

Returns

An array containing the integration point data.

Return type

Array

Example

To get the data for the 3rd integration point for initial stress shell iss:

```
var data = iss.GetIntegrationPoint(2);
```

GetParameter(prop*[string]*)

Description

Checks if a StressShell property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the StressShell.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

initial stress shell property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if StressShell property iss.example is a parameter:

```
Options.property_parameter_names = true;
if (iss.GetParameter(iss.example) ) do_something...
Options.property_parameter_names = false;
```

To check if StressShell property iss.example is a parameter by using the GetParameter method:

```
if (iss.ViewParameters().GetParameter(iss.example) ) do_something...
```

GetThermalIntegrationPoint(index*[integer]*)

Description

Returns the thermal data for a specific integration point as an array. For each integration point there will be nthhsv values. There are nthint integration points.

Arguments

- **index** (integer)

Index you want the integration point data for. **Note that indices start at 0**.

Returns

An array containing the integration point data.

Return type

Array

Example

To get the data for the 3rd thermal integration point for initial stress shell iss:

```
var data = iss.GetThermalIntegrationPoint(2);
```

Keyword()

Description

Returns the keyword for this initial stress shell (*INITIAL_STRESS_SHELL). **Note that a carriage return is not added**. See also StressShell.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for stress_shell i:

```
var key = i.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the initial stress shell. **Note that a carriage return is not added**. See also StressShell.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for stress_shell i:

```
var cards = i.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last initial stress shell in the model.

Arguments

- **Model** (Model)

Model to get last initial stress shell in

Returns

StressShell object (or null if there are no initial stress shells in the model).

Return type

StressShell

Example

To get the last initial stress shell in model m:

```
var iss = StressShell.Last(m);
```

Next()

Description

Returns the next initial stress shell in the model.

Arguments

No arguments

Returns

StressShell object (or null if there are no more initial stress shells in the model).

Return type

StressShell

Example

To get the initial stress shell in model m after initial stress shell iss:

```
var iss = iss.Next();
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a initial stress shell.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only initial stress shells from that model can be picked. If the argument is a Flag then only initial stress shells that are flagged with *limit* can be selected. If omitted, or null, any initial stress shells from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

StressShell object (or null if not picked)

Return type

StressShell

Example

To pick a initial stress shell from model m giving the prompt 'Pick initial stress shell from screen':

```
var iss = StressShell.Pick('Pick initial stress shell from screen', m);
```

Previous()

Description

Returns the previous initial stress shell in the model.

Arguments

No arguments

Returns

StressShell object (or null if there are no more initial stress shells in the model).

Return type

StressShell

Example

To get the initial stress shell in model m before initial stress shell iss:

```
var iss = iss.Previous();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select initial stress shells using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting initial stress shells

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only initial stress shells from that model can be selected. If the argument is a Flag then only initial stress shells that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any initial stress shells can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of initial stress shells selected or null if menu cancelled

Return type

Number

Example

To select initial stress shells from model m, flagging those selected with flag f, giving the prompt 'Select initial stress shells':

```
StressShell.Select(f, 'Select initial stress shells', m);
```

To select initial stress shells, flagging those selected with flag f but limiting selection to initial stress shells flagged with flag l, giving the prompt 'Select initial stress shells':

```
StressShell.Select(f, 'Select initial stress shells', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the initial stress shell.

Arguments

- **flag** (Flag)

Flag to set on the initial stress shell

Returns

No return value

Example

To set flag f for initial stress shell iss:

```
iss.SetFlag(f);
```

SetIntegrationPoint(index*[integer]*, data*[Array of data]*)

Description

Set the data for a specific integration point. For each integration point there will be 8 + nhisv + (6 x ntensr) values. There are nplane x nthick integration points.

Arguments

- **index** (integer)

Index you want the integration point data for. **Note that indices start at 0**.

- **data** (Array of data)

Array containing the integration point data. The array length should be 8 + nhisv + (6 x ntensr).

Returns

No return value.

Example

To set the 3rd integration point data for initial stress shell iss to the values in array adata:

```
iss.SetIntegrationPoint(2, adata);
```

SetThermalIntegrationPoint(index*[integer]*, data*[Array of data]*)

Description

Set the thermal data for a specific integration point. For each integration point there will be nthhsv values. There are nthint thermal integration points.

Arguments

- **index** (integer)

Index you want the thermal integration point data for. **Note that indices start at 0**.

- **data** (Array of data)

Array containing the thermal integration point data. The array length should be nthhsv.

Returns

No return value.

Example

To set the 3rd thermal integration point data for initial stress shell iss to the values in array adata:

```
iss.SetThermalIntegrationPoint(2, adata);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the initial stress shell. The initial stress shell will be sketched until you either call StressShell.Unsketch(), StressShell.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the initial stress shell is sketched. If omitted redraw is true. If you want to sketch several initial stress shells and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch initial stress shell iss:

```
iss.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged initial stress shells in the model. The initial stress shells will be sketched until you either call StressShell.Unsketch(), StressShell.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged initial stress shells will be sketched in

- **flag** (Flag)

Flag set on the initial stress shells that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the initial stress shells are sketched. If omitted redraw is true. If you want to sketch flagged initial stress shells several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all initial stress shells flagged with flag in model m:

```
StressShell.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of initial stress shells in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing initial stress shells should be counted. If false or omitted referenced but undefined initial stress shells will also be included in the total.

Returns

number of initial stress shells

Return type

Number

Example

To get the total number of initial stress shells in model m:

```
var total = StressShell.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the initial stress shells in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all initial stress shells will be unset in

- **flag** (Flag)

Flag to unset on the initial stress shells

Returns

No return value

Example

To unset the flag f on all the initial stress shells in model m:

```
StressShell.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the initial stress shell.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the initial stress shell is unsketched. If omitted redraw is true. If you want to unsketch several initial stress shells and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch initial stress shell iss:

```
iss.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all initial stress shells.

Arguments

- **Model** (Model)

Model that all initial stress shells will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the initial stress shells are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all initial stress shells in model m:

```
StressShell.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged initial stress shells in the model.

Arguments

- **Model** (Model)

Model that all initial stress shells will be unsketched in

- **flag** (Flag)

Flag set on the initial stress shells that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the initial stress shells are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all initial stress shells flagged with flag in model m:

```
StressShell.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

StressShell object.

Return type

StressShell

Example

To check if StressShell property iss.example is a parameter by using the StressShell.GetParameter() method:

```
if (iss.ViewParameters().GetParameter(iss.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for initial stress shell with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description

autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The initial stress shell object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for initial stress shell iss:

```
iss.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for initial stress shell iss:

```
iss.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for initial stress shell. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for initial stress shell iss:

```
iss.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this initial stress shell.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for initial stress shell iss:

```
var xrefs = iss.Xrefs();
```

---

toString()

Description

Creates a string containing the initial stress shell data in keyword format. Note that this contains the keyword header and the keyword cards. See also StressShell.Keyword() and StressShell.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for stress_shell i in keyword format

```
var s = i.toString();
```

StressSolid class

The StressSolid class gives you access to define initial stress solid cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- First(Model*[Model]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetIntegrationPoint(index*[integer]*)
- GetParameter(prop*[string]*)
- GetThermalIntegrationPoint(index*[integer]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- SetIntegrationPoint(index*[integer]*, data*[Array of data]*)
- SetThermalIntegrationPoint(index*[integer]*, data*[Array of data]*)
- Sketch(redraw (optional)*[boolean]*)
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

StressSolid constants

Name	Description
StressSolid.SET	Initial is *INITIAL_STRESS_SOLID_SET.
StressSolid.SOLID	Initial is *INITIAL_STRESS_SOLID.

StressSolid properties

Name	Type	Description
eid	integer	Solid Element ID or solid set ID
exists (read only)	logical	true if initial stress solid exists, false if referred to but not defined.
ialegp	integer	*ALE_MULTI-MATERIAL_GROUP or *ALE_STRUCTURED_MULTI-MATERIAL_GROUP id.
include	integer	The Include file number that the initial stress solid is in.
iveflg	integer	Initial volume energy flag (only used if large is TRUE). Valid values are 0, 1 and 2 only.
large	logical	true if large format, false otherwise.
model (read only)	integer	The Model number that the initial stress solid is in.
nhisv	integer	Number of additional history variables (only used if large is TRUE).
nint	integer	Number of integration points (should correspond to the solid element formulation). Valid values for hexadral solid elements are 1, 8 or 14. Valid values for tetrahedral elements are 1, 4 or 5. Valid values for pentraderal elements are 1 or 2.
nthhsv	integer	Number of thermal history variables per thermal integration point (only used if large is TRUE).
nthint	integer	Number of thermal integration points (only used if large is TRUE).
type	constant	The Intial stress solid type. Can be StressSolid.SOLID or StressSolid.SET.

Detailed Description

The StressSolid class allows you to create, modify, edit and manipulate *INITIAL_STRESS_SOLID cards. See the documentation below for more details.

Constructor

new StressSolid(Model*[Model]*, type*[constant]*, eid*[integer]*, nint*[integer]*, nhisv*[integer]*, large*[boolean]*, iveflg*[integer]*)

Description

Create a new StressSolid object.

Arguments

- **Model** (Model)

Model that stress_solid will be created in

- **type** (constant)

Specify the type of initial stress solid (Can be StressSolid.SOLID or StressSolid.SET)

- **eid** (integer)

Solid Element ID or solid set ID

- **nint** (integer)

Number of integration points (should correspond to the solid element formulation). Valid values for hexadral solid elements are 1, 8 or 14. Valid values for tetrahedral elements are 1, 4 or 5. Valid values for pentraderal elements are 1 or 2.

- **nhisv** (integer)

Number of additional history variables (only used if large is TRUE).

- **large** (boolean)

true if large format, false otherwise.

- **iveflg** (integer)

Initial volume energy flag (only used if large is TRUE). Valid values are 0, 1 and 2 only.

Returns

StressSolid object

Return type

StressSolid

Example

To create a new stress_solid in model m, of type SET with SOLID_SET id as 1, number of integration points as 3 and number of history variables as 4.

```
var s = new StressSolid(m, StressSolid.SET, 1, 3, 4, true);
```

Details of functions

AssociateComment(Comment[*Comment*])

Description

Associates a comment with a initial stress solid.

Arguments

- **Comment** (Comment)

Comment that will be attached to the initial stress solid

Returns

No return value

Example

To associate comment c to the initial stress solid iso:

```
iso.AssociateComment(c);
```

ClearFlag(flag[*Flag*])

Description

Clears a flag on the initial stress solid.

Arguments

- **flag** (Flag)

Flag to clear on the initial stress solid

Returns

No return value

Example

To clear flag f for initial stress solid iso:

```
iso.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the initial stress solid. The target include of the copied initial stress solid can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

StressSolid object

Return type

StressSolid

Example

To copy initial stress solid iso into initial stress solid z:

```
var z = iso.Copy();
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a initial stress solid.

Arguments

- **Comment** (Comment)

Comment that will be detached from the initial stress solid

Returns

No return value

Example

To detach comment c from the initial stress solid iso:

```
iso.DetachComment(c);
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for initial stress solid with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The initial stress solid object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for initial stress solid iso:

```
iso.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for initial stress solid iso:

```
iso.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for initial stress solid. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for initial stress solid iso:

```
iso.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first initial stress solid in the model.

Arguments

- **Model** (Model)

Model to get first initial stress solid in

Returns

StressSolid object (or null if there are no initial stress solids in the model).

Return type

StressSolid

Example

To get the first initial stress solid in model m:

```
var iso = StressSolid.First(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the initial stress solids in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all initial stress solids will be flagged in

- **flag** (Flag)

Flag to set on the initial stress solids

Returns

No return value

Example

To flag all of the initial stress solids with flag f in model m:

```
StressSolid.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the initial stress solid is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the initial stress solid

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if initial stress solid iso has flag f set on it:

```
if (iso.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each initial stress solid in the model.
Note that ForEach has been designed to make looping over initial stress solids as fast as possible and so has some limitations.
Firstly, a single temporary StressSolid object is created and on each function call it is updated with the current initial stress solid data. This means that you should not try to store the StressSolid object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new initial stress solids inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all initial stress solids are in

- **func** (function)

Function to call for each initial stress solid

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the initial stress solids in model m:

```
StressSolid.ForEach(m, test);
function test(iso)
{
// iso is StressSolid object
}
```

To call function test for all of the initial stress solids in model m with optional object:

```
var data = { x:0, y:0 };
StressSolid.ForEach(m, test, data);
function test(iso, extra)
{
// iso is StressSolid object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of StressSolid objects or properties for all of the initial stress solids in a model in PRIMER. If the optional property argument is not given then an array of StressSolid objects is returned. If the property argument is given, that property value for each initial stress solid is returned in the array instead of a StressSolid object

Arguments

- **Model** (Model)

Model to get initial stress solids from

- **property (optional)** (string)

Name for property to get for all initial stress solids in the model

Returns

Array of StressSolid objects or properties

Return type

Array

Example

To make an array of StressSolid objects for all of the initial stress solids in model m:

```
var a = StressSolid.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each initial stress solid in model m:

```
var a = StressSolid.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a initial stress solid.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the initial stress solid iso:

```
var comm_array = iso.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of StressSolid objects for all of the flagged initial stress solids in a model in PRIMER If the optional property argument is not given then an array of StressSolid objects is returned. If the property argument is given, then that property value for each initial stress solid is returned in the array instead of a StressSolid object

Arguments

- **Model** (Model)

Model to get initial stress solids from

- **flag** (Flag)

Flag set on the initial stress solids that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged initial stress solids in the model

Returns

Array of StressSolid objects or properties

Return type

Array

Example

To make an array of StressSolid objects for all of the initial stress solids in model m flagged with f:

```
var iso = StressSolid.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the initial stress solids in model m flagged with f:

```
var a = StressSolid.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the StressSolid object for a initial stress solid ID.

Arguments

- **Model** (Model)

Model to find the initial stress solid in

- **number** (integer)

number of the initial stress solid you want the StressSolid object for

Returns

StressSolid object (or null if initial stress solid does not exist).

Return type

StressSolid

Example

To get the StressSolid object for initial stress solid 100 in model m

```
var iso = StressSolid.GetFromID(m, 100);
```

GetIntegrationPoint(index*[integer]*)

Description

Returns the data for a specific integration point as an array. For each integration point there will be 7 values if large is FALSE. For each integration point there will be (7 + nhisv) values if large is TRUE. There are nint integration points.

Arguments

- **index** (integer)

Index you want the integration point data for. **Note that indices start at 0**.

Returns

An array containing the integration point data.

Return type

Array

Example

To get the data for the 3rd integration point for initial stress solid iso:

```
var data = iso.GetIntegrationPoint(2);
```

GetParameter(prop*[string]*)

Description

Checks if a StressSolid property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the StressSolid.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

initial stress solid property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if StressSolid property iso.example is a parameter:

```
Options.property_parameter_names = true;
if (iso.GetParameter(iso.example) ) do_something...
Options.property_parameter_names = false;
```

To check if StressSolid property iso.example is a parameter by using the GetParameter method:

```
if (iso.ViewParameters().GetParameter(iso.example) ) do_something...
```

GetThermalIntegrationPoint(index*[integer]*)

Description

Returns the thermal data for a specific integration point as an array. For each integration point there will be nthhsv values. There are nthint integration points.

Arguments

- **index** (integer)

Index you want the integration point data for. **Note that indices start at 0**.

Returns

An array containing the integration point data.

Return type

Array

Example

To get the data for the 3rd thermal integration point for initial stress solid iso:

```
var data = iso.GetThermalIntegrationPoint(2);
```

Keyword()

Description

Returns the keyword for this initial stress solid (*INITIAL_STRESS_SOLID). **Note that a carriage return is not added**. See also StressSolid.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for stress_solid i:

```
var key = i.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the initial stress solid. **Note that a carriage return is not added**. See also StressSolid.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for stress_solid i:

```
var cards = i.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last initial stress solid in the model.

Arguments

- **Model** (Model)

Model to get last initial stress solid in

Returns

StressSolid object (or null if there are no initial stress solids in the model).

Return type

StressSolid

Example

To get the last initial stress solid in model m:

```
var iso = StressSolid.Last(m);
```

Next()

Description

Returns the next initial stress solid in the model.

Arguments

No arguments

Returns

StressSolid object (or null if there are no more initial stress solids in the model).

Return type

StressSolid

Example

To get the initial stress solid in model m after initial stress solid iso:

```
var iso = iso.Next();
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a initial stress solid.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only initial stress solids from that model can be picked. If the argument is a Flag then only initial stress solids that are flagged with *limit* can be selected. If omitted, or null, any initial stress solids from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

StressSolid object (or null if not picked)

Return type

StressSolid

Example

To pick a initial stress solid from model m giving the prompt 'Pick initial stress solid from screen':

```
var iso = StressSolid.Pick('Pick initial stress solid from screen', m);
```

Previous()

Description

Returns the previous initial stress solid in the model.

Arguments

No arguments

Returns

StressSolid object (or null if there are no more initial stress solids in the model).

Return type

StressSolid

Example

To get the initial stress solid in model m before initial stress solid iso:

```
var iso = iso.Previous();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select initial stress solids using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting initial stress solids

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only initial stress solids from that model can be selected. If the argument is a Flag then only initial stress solids that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any initial stress solids can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of initial stress solids selected or null if menu cancelled

Return type

Number

Example

To select initial stress solids from model m, flagging those selected with flag f, giving the prompt 'Select initial stress solids':

```
StressSolid.Select(f, 'Select initial stress solids', m);
```

To select initial stress solids, flagging those selected with flag f but limiting selection to initial stress solids flagged with flag l, giving the prompt 'Select initial stress solids':

```
StressSolid.Select(f, 'Select initial stress solids', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the initial stress solid.

Arguments

- **flag** (Flag)

Flag to set on the initial stress solid

Returns

No return value

Example

To set flag f for initial stress solid iso:

```
iso.SetFlag(f);
```

SetIntegrationPoint(index*[integer]*, data*[Array of data]*)

Description

Set the data for a specific integration point. For each integration point there will be 7 values if large is FALSE. For each integration point there will be (7 + nhisv) values if large is TRUE. There are nint integration points.

Arguments

- **index** (integer)

Index you want the integration point data for. **Note that indices start at 0**.

- **data** (Array of data)

Array containing the integration point data. The array length should be 7 if large is FALSE. The array length should be (7 + nhisv) if large is TRUE.

Returns

No return value.

Example

To set the 3rd integration point data for initial stress solid iso to the values in array adata:

```
iso.SetIntegrationPoint(2, adata);
```

SetThermalIntegrationPoint(index*[integer]*, data*[Array of data]*)

Description

Set the thermal data for a specific integration point. For each integration point there will be nthhsv values. There are nthint thermal integration points.

Arguments

- **index** (integer)

Index you want the thermal integration point data for. **Note that indices start at 0**.

- **data** (Array of data)

Array containing the thermal integration point data. The array length should be nthhsv.

Returns

No return value.

Example

To set the 3rd thermal integration point data for initial stress solid iso to the values in array adata:

```
iso.SetThermalIntegrationPoint(2, adata);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the initial stress solid. The initial stress solid will be sketched until you either call StressSolid.Unsketch(), StressSolid.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the initial stress solid is sketched. If omitted redraw is true. If you want to sketch several initial stress solids and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch initial stress solid iso:

```
iso.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged initial stress solids in the model. The initial stress solids will be sketched until you either call StressSolid.Unsketch(), StressSolid.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged initial stress solids will be sketched in

- **flag** (Flag)

Flag set on the initial stress solids that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the initial stress solids are sketched. If omitted redraw is true. If you want to sketch flagged initial stress solids several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all initial stress solids flagged with flag in model m:

```
StressSolid.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of initial stress solids in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing initial stress solids should be counted. If false or omitted referenced but undefined initial stress solids will also be included in the total.

Returns

number of initial stress solids

Return type

Number

Example

To get the total number of initial stress solids in model m:

```
var total = StressSolid.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the initial stress solids in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all initial stress solids will be unset in

- **flag** (Flag)

Flag to unset on the initial stress solids

Returns

No return value

Example

To unset the flag f on all the initial stress solids in model m:

```
StressSolid.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the initial stress solid.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the initial stress solid is unsketched. If omitted redraw is true. If you want to unsketch several initial stress solids and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch initial stress solid iso:

```
iso.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all initial stress solids.

Arguments

- **Model** (Model)

Model that all initial stress solids will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the initial stress solids are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all initial stress solids in model m:

```
StressSolid.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged initial stress solids in the model.

Arguments

- **Model** (Model)

Model that all initial stress solids will be unsketched in

- **flag** (Flag)

Flag set on the initial stress solids that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the initial stress solids are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all initial stress solids flagged with flag in model m:

```
StressSolid.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

StressSolid object.

Return type

StressSolid

Example

To check if StressSolid property iso.example is a parameter by using the StressSolid.GetParameter() method:

```
if (iso.ViewParameters().GetParameter(iso.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for initial stress solid with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description

autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The initial stress solid object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for initial stress solid iso:

```
iso.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for initial stress solid iso:

```
iso.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for initial stress solid. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for initial stress solid iso:

```
iso.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this initial stress solid.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for initial stress solid iso:

```
var xrefs = iso.Xrefs();
```

---

toString()

Description

Creates a string containing the initial stress solid data in keyword format. Note that this contains the keyword header and the keyword cards. See also StressSolid.Keyword() and StressSolid.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for stress_solid i in keyword format

```
var s = i.toString();
```

Velocity class

The Velocity class gives you access to define initial velocity cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- First(Model*[Model]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

Velocity properties

Name	Type	Description

boxid	integer	Define box containing nodes
exists (read only)	logical	true if initial velocity exists, false if referred to but not defined.
icid	integer	Local coordinate system
include	integer	The Include file number that the initial velocity is in.
irigid	integer	IRIGID flag
model (read only)	integer	The Model number that the initial velocity is in.
nsid	integer	Set Node set ID
nsidex	integer	Set Exempted Node set ID
vx	real	Initial velocity in X direction
vxe	real	Initial velocity in X direction of exempted nodes
vxr	real	Initial rotational velocity about X axis
vxre	real	Initial rotational velocity about X axis of exempted nodes
vy	real	Initial velocity in Y direction
vye	real	Initial velocity in Y direction of exempted nodes
vyr	real	Initial rotational velocity about Y axis
vyre	real	Initial rotational velocity about Y axis of exempted nodes
vz	real	Initial velocity in Z direction
vze	real	Initial velocity in Z direction of exempted nodes
vzr	real	Initial rotational velocity about Z axis
vzre	real	Initial rotational velocity about Z axis of exempted nodes

Detailed Description

The Velocity class allows you to create, modify, edit and manipulate velocity cards. See the documentation below for more details.

Constructor

new Velocity(Model*[Model]*, nsid*[integer]*, vx*[real]*, vy*[real]*, vz*[real]*, vxr*[real]*, vyr*[real]*, vzr*[real]*, boxid (optional)*[integer]*, irigid (optional)*[integer]*, nsidex (optional)*[integer]*, vxe (optional)*[real]*, vye (optional)*[real]*, vze (optional)*[real]*, vxre (optional)*[real]*, vyre (optional)*[real]*, vzre (optional)*[real]*, icid (optional)*[real]*)

Description

Create a new Velocity object.

Arguments

- **Model** (Model)

Model that velocity will be created in

- **nsid** (integer)

Set Node set ID

- **vx** (real)

Initial velocity in X direction

- **vy** (real)

Initial velocity in Y direction

- **vz** (real)

Initial velocity in Z direction

- **vxr** (real)

Initial rotational velocity about X axis

- **vyr** (real)

Initial rotational velocity about Y axis

- **vzr** (real)

Initial rotational velocity about Z axis

- **boxid (optional)** (integer)

Define box containing nodes

- **irigid (optional)** (integer)

IRIGID flag

- **nsidex (optional)** (integer)

Set Exempted Node set ID

- **vxe (optional)** (real)

Initial velocity in X direction of exempted nodes

- **vye (optional)** (real)

Initial velocity in Y direction of exempted nodes

- **vze (optional)** (real)

Initial velocity in Z direction of exempted nodes

- **vxre (optional)** (real)

Initial rotational velocity about X axis of exempted nodes

- **vyre (optional)** (real)

Initial rotational velocity about Y axis of exempted nodes

- **vzre (optional)** (real)

Initial rotational velocity about Z axis of exempted nodes

- **icid (optional)** (real)

Local coordinate system nodes

Returns

Velocity object

Return type

Velocity

Example

To create a new velocity in model m

```
var s = new Velocity(m, 1, 2.4, 3.7, 7.9, 0.0, 0.0, 0.0);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a initial velocity.

Arguments

- **Comment** (Comment)

Comment that will be attached to the initial velocity

Returns

No return value

Example

To associate comment c to the initial velocity v:

```
v.AssociateComment(c);
```

Blank()

Description

Blanks the initial velocity

Arguments

No arguments

Returns

No return value

Example

To blank initial velocity v:

```
v.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the initial velocities in the model.

Arguments

- **Model** (Model)

Model that all initial velocities will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the initial velocities in model m:

```
Velocity.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged initial velocities in the model.

Arguments

- **Model** (Model)

Model that all the flagged initial velocities will be blanked in

- **flag** (Flag)

Flag set on the initial velocities that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the initial velocities in model m flagged with f:

```
Velocity.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the initial velocity is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if initial velocity v is blanked:

```
if (v.Blanked() ) do_something...
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the initial velocity.

Arguments

- **flag** (Flag)

Flag to clear on the initial velocity

Returns

No return value

Example

To clear flag f for initial velocity v:

```
v.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the initial velocity. The target include of the copied initial velocity can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

Velocity object

Return type

Velocity

Example

To copy initial velocity v into initial velocity z:

```
var z = v.Copy();
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a initial velocity.

Arguments

- **Comment** (Comment)

Comment that will be detached from the initial velocity

Returns

No return value

Example

To detach comment c from the initial velocity v:

```
v.DetachComment(c);
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for initial velocity with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The initial velocity object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for initial velocity v:

```
v.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for initial velocity v:

```
v.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for initial velocity. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for initial velocity v:

```
v.Error("My custom error");
```

First(Model[*Model*]) [static]

Description

Returns the first initial velocity in the model.

Arguments

- **Model** (Model)

Model to get first initial velocity in

Returns

Velocity object (or null if there are no initial velocities in the model).

Return type

Velocity

Example

To get the first initial velocity in model m:

```
var v = Velocity.First(m);
```

FlagAll(Model[*Model*], flag[*Flag*]) [static]

Description

Flags all of the initial velocities in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all initial velocities will be flagged in

- **flag** (Flag)

Flag to set on the initial velocities

Returns

No return value

Example

To flag all of the initial velocities with flag f in model m:

```
Velocity.FlagAll(m, f);
```

Flagged(flag[*Flag*])

Description

Checks if the initial velocity is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the initial velocity

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if initial velocity v has flag f set on it:

```
if (v.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each initial velocity in the model.
Note that ForEach has been designed to make looping over initial velocities as fast as possible and so has some limitations.
Firstly, a single temporary Velocity object is created and on each function call it is updated with the current initial velocity data. This means that you should not try to store the Velocity object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new initial velocities inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all initial velocities are in

- **func** (function)

Function to call for each initial velocity

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the initial velocities in model m:

```
Velocity.ForEach(m, test);
function test(v)
{
// v is Velocity object
}
```

To call function test for all of the initial velocities in model m with optional object:

```
var data = { x:0, y:0 };
Velocity.ForEach(m, test, data);
function test(v, extra)
{
// v is Velocity object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Velocity objects or properties for all of the initial velocities in a model in PRIMER. If the optional property argument is not given then an array of Velocity objects is returned. If the property argument is given, that property value for each initial velocity is returned in the array instead of a Velocity object

Arguments

- **Model** (Model)

Model to get initial velocities from

- **property (optional)** (string)

Name for property to get for all initial velocities in the model

Returns

Array of Velocity objects or properties

Return type

Array

Example

To make an array of Velocity objects for all of the initial velocities in model m:

```
var a = Velocity.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each initial velocity in model m:

```
var a = Velocity.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a initial velocity.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the initial velocity v:

```
var comm_array = v.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of Velocity objects for all of the flagged initial velocities in a model in PRIMER If the optional property argument is not given then an array of Velocity objects is returned. If the property argument is given, then that property value for each initial velocity is returned in the array instead of a Velocity object

Arguments

- **Model** (Model)

Model to get initial velocities from

- **flag** (Flag)

Flag set on the initial velocities that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged initial velocities in the model

Returns

Array of Velocity objects or properties

Return type

Array

Example

To make an array of Velocity objects for all of the initial velocities in model m flagged with f:

```
var v = Velocity.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the initial velocities in model m flagged with f:

```
var a = Velocity.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the Velocity object for a initial velocity ID.

Arguments

- **Model** (Model)

Model to find the initial velocity in

- **number** (integer)

number of the initial velocity you want the Velocity object for

Returns

Velocity object (or null if initial velocity does not exist).

Return type

Velocity

Example

To get the Velocity object for initial velocity 100 in model m

```
var v = Velocity.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a Velocity property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the Velocity.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

initial velocity property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if Velocity property v.example is a parameter:

```
Options.property_parameter_names = true;
if (v.GetParameter(v.example) ) do_something...
Options.property_parameter_names = false;
```

To check if Velocity property v.example is a parameter by using the GetParameter method:

```
if (v.ViewParameters().GetParameter(v.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this initial velocity (*INITIAL_VELOCITY). **Note that a carriage return is not added**. See also Velocity.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for velocity i:

```
var key = i.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the initial velocity. **Note that a carriage return is not added**. See also Velocity.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for velocity i:

```
var cards = i.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last initial velocity in the model.

Arguments

- **Model** (Model)

Model to get last initial velocity in

Returns

Velocity object (or null if there are no initial velocities in the model).

Return type

Velocity

Example

To get the last initial velocity in model m:

```
var v = Velocity.Last(m);
```

Next()

Description

Returns the next initial velocity in the model.

Arguments

No arguments

Returns

Velocity object (or null if there are no more initial velocities in the model).

Return type

Velocity

Example

To get the initial velocity in model m after initial velocity v:

```
var v = v.Next();
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a initial velocity.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only initial velocities from that model can be picked. If the argument is a Flag then only initial velocities that are flagged with *limit* can be selected. If omitted, or null, any initial velocities from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

Velocity object (or null if not picked)

Return type

Velocity

Example

To pick a initial velocity from model m giving the prompt 'Pick initial velocity from screen':

```
var v = Velocity.Pick('Pick initial velocity from screen', m);
```

Previous()

Description

Returns the previous initial velocity in the model.

Arguments

No arguments

Returns

Velocity object (or null if there are no more initial velocities in the model).

Return type

Velocity

Example

To get the initial velocity in model m before initial velocity v:

```
var v = v.Previous();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select initial velocities using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting initial velocities

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only initial velocities from that model can be selected. If the argument is a Flag then only initial velocities that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any initial velocities can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of initial velocities selected or null if menu cancelled

Return type

Number

Example

To select initial velocities from model m, flagging those selected with flag f, giving the prompt 'Select initial velocities':

```
Velocity.Select(f, 'Select initial velocities', m);
```

To select initial velocities, flagging those selected with flag f but limiting selection to initial velocities flagged with flag l, giving the prompt 'Select initial velocities':

```
Velocity.Select(f, 'Select initial velocities', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the initial velocity.

Arguments

- **flag** (Flag)

Flag to set on the initial velocity

Returns

No return value

Example

To set flag f for initial velocity v:

```
v.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the initial velocity. The initial velocity will be sketched until you either call Velocity.Unsketch(), Velocity.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the initial velocity is sketched. If omitted redraw is true. If you want to sketch several initial velocities and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch initial velocity v:

```
v.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged initial velocities in the model. The initial velocities will be sketched until you either call Velocity.Unsketch(), Velocity.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged initial velocities will be sketched in

- **flag** (Flag)

Flag set on the initial velocities that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the initial velocities are sketched. If omitted redraw is true. If you want to sketch flagged initial velocities several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all initial velocities flagged with flag in model m:

```
Velocity.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of initial velocities in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing initial velocities should be counted. If false or omitted referenced but undefined initial velocities will also be included in the total.

Returns

number of initial velocities

Return type

Number

Example

To get the total number of initial velocities in model m:

```
var total = Velocity.Total(m);
```

Unblank()

Description

Unblanks the initial velocity

Arguments

No arguments

Returns

No return value

Example

To unblank initial velocity v:

```
v.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the initial velocities in the model.

Arguments

- **Model** (Model)

Model that all initial velocities will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the initial velocities in model m:

```
Velocity.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged initial velocities in the model.

Arguments

- **Model** (Model)

Model that the flagged initial velocities will be unblanked in

- **flag** (Flag)

Flag set on the initial velocities that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the initial velocities in model m flagged with f:

```
Velocity.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the initial velocities in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all initial velocities will be unset in

- **flag** (Flag)

Flag to unset on the initial velocities

Returns

No return value

Example

To unset the flag f on all the initial velocities in model m:

```
Velocity.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the initial velocity.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the initial velocity is unsketched. If omitted redraw is true. If you want to unsketch several initial velocities and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch initial velocity v:

```
v.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all initial velocities.

Arguments

- **Model** (Model)

Model that all initial velocities will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the initial velocities are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all initial velocities in model m:

```
Velocity.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged initial velocities in the model.

Arguments

- **Model** (Model)

Model that all initial velocities will be unsketched in

- **flag** (Flag)

Flag set on the initial velocities that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the initial velocities are unsketched. If omitted redraw is true. If you want to

unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all initial velocities flagged with flag in model m:

```
Velocity.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

Velocity object.

Return type

Velocity

Example

To check if Velocity property v.example is a parameter by using the Velocity.GetParameter() method:

```
if (v.ViewParameters().GetParameter(v.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for initial velocity with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The initial velocity object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...

autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for initial velocity v:

```
v.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for initial velocity v:

```
v.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for initial velocity. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for initial velocity v:

```
v.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this initial velocity.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for initial velocity v:

```
var xrefs = v.Xrefs();
```

---

toString()

Description

Creates a string containing the initial velocity data in keyword format. Note that this contains the keyword header and the keyword cards. See also Velocity.Keyword() and Velocity.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for velocity i in keyword format

```
var s = i.toString();
```

VelocityGeneration class

The VelocityGeneration class gives you access to define initial velocity generation cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model *[Model]*, redraw (optional) *[boolean]*)
- BlankFlagged(Model *[Model]*, flag *[Flag]*, redraw (optional) *[boolean]*)
- Create(Model *[Model]*, modal (optional) *[boolean]*)
- First(Model *[Model]*)
- FlagAll(Model *[Model]*, flag *[Flag]*)
- ForEach(Model *[Model]*, func *[function]*, extra (optional) *[any]*)
- GetAll(Model *[Model]*, property (optional) *[string]*)
- GetFlagged(Model *[Model]*, flag *[Flag]*, property (optional) *[string]*)
- GetFromID(Model *[Model]*, number *[integer]*)
- Last(Model *[Model]*)
- Pick(prompt *[string]*, limit (optional) *[Model or Flag]*, modal (optional) *[boolean]*, button text (optional) *[string]*)
- Select(flag *[Flag]*, prompt *[string]*, limit (optional) *[Model or Flag]*, modal (optional) *[boolean]*)
- SketchFlagged(Model *[Model]*, flag *[Flag]*, redraw (optional) *[boolean]*)
- Total(Model *[Model]*, exists (optional) *[boolean]*)
- UnblankAll(Model *[Model]*, redraw (optional) *[boolean]*)
- UnblankFlagged(Model *[Model]*, flag *[Flag]*, redraw (optional) *[boolean]*)
- UnflagAll(Model *[Model]*, flag *[Flag]*)
- UnsketchAll(Model *[Model]*, redraw (optional) *[boolean]*)
- UnsketchFlagged(Model *[Model]*, flag *[Flag]*, redraw (optional) *[boolean]*)

Member functions

- AssociateComment(Comment *[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional) *[boolean]*)
- ClearFlag(flag *[Flag]*)
- Copy(range (optional) *[boolean]*)
- DetachComment(Comment *[Comment]*)
- Edit(modal (optional) *[boolean]*)
- Error(message *[string]*, options (optional) *[object]*)
- Error(message *[string]*, details (optional) *[string]*) [deprecated]
- Flagged(flag *[Flag]*)
- GetComments()
- GetParameter(prop *[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag *[Flag]*)
- Sketch(redraw (optional) *[boolean]*)
- Unblank()
- Unsketch(redraw (optional) *[boolean]*)
- ViewParameters()
- Warning(message *[string]*, options (optional) *[object]*)
- Warning(message *[string]*, details (optional) *[string]*) [deprecated]
- Xrefs()
- toString()

VelocityGeneration constants

Name	Description
VelocityGeneration.NODE_SET	ID is a NODE_SET
VelocityGeneration.PART	ID is a PART
VelocityGeneration.PART_SET	ID is a PART_SET

VelocityGeneration properties

Name	Type	Description
exists (read only)	logical	true if initial velocity generation exists, false if referred to but not defined.
icid	integer	Local coordinate system
id	integer	Set Part ID, Part set ID or Node set ID
include	integer	The Include file number that the initial velocity generation is in.
irigid	integer	Overide part inertia flag
ivatn	integer	Tracked parts flag
model (read only)	integer	The Model number that the initial velocity generation is in.
nx	real	x-direction cosine
ny	real	y-direction cosine
nz	real	z-direction cosine
omega	real	Angular velocity about the rotational axis
phase	integer	Dynamic relaxation flag
type	constant	Specify the type of Velocity generation (Can be VelocityGeneration.PART_SET or VelocityGeneration.PART or VelocityGeneration.NODE_SET)
vx	real	Initial translational velocity in X direction
vy	real	Initial translational velocity in Y direction
vz	real	Initial translational velocity in Z direction
xc	real	x-coordinate on rotational axis
yc	real	y-coordinate on rotational axis
zc	real	z-coordinate on rotational axis

Detailed Description

The VelocityGeneration class allows you to create, modify, edit and manipulate velocity cards. See the documentation below for more details.

Constructor

new VelocityGeneration(Model*[Model]*, type*[constant]*, id*[integer]*, omega*[real]*, vx*[real]*, vy*[real]*, vz*[real]*, ivatn*[integer]*, xc*[real]*, yc*[real]*, zc*[real]*, nx*[real]*, ny*[real]*, nz*[real]*, phase*[integer]*, irigid*[integer]*, icid*[integer]*)

Description

Create a new VelocityGeneration object.

Arguments

- **Model** (Model)

Model that velocity will be created in

- **type** (constant)

Specify the type of Velocity generation (Can be VelocityGeneration.PART_SET or VelocityGeneration.PART or VelocityGeneration.NODE_SET)

- **id** (integer)

Set Part ID, Part set ID or Node set ID

- **omega** (real)

Angular velocity about the rotational axis

- **vx** (real)

Initial translational velocity in X direction

- **vy** (real)

Initial translational velocity in Y direction

- **vz** (real)

Initial translational velocity in Z direction

- **ivatn** (integer)

Tracked parts flag

- **xc** (real)

x-coordinate on rotational axis

- **yc** (real)

y-coordinate on rotational axis

- **zc** (real)

z-coordinate on rotational axis

- **nx** (real)

x-direction cosine

- **ny** (real)

y-direction cosine

- **nz** (real)

z-direction cosine

- **phase** (integer)

Dynamic relaxation flag

- **irigid** (integer)

Overide part inertia flag

- **icid** (integer)

Local coordinate system

Returns

VelocityGeneration object

Return type

VelocityGeneration

Example

To create a new velocity in model m

```
var s = new VelocityGeneration(m, VelocityGeneration.PART, 500, 3.4, 2.4, 3.7,
7.9, 0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1, 0);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a initial velocity generation.

Arguments

- **Comment** (Comment)

Comment that will be attached to the initial velocity generation

Returns

No return value

Example

To associate comment c to the initial velocity generation ivg:

```
ivg.AssociateComment(c);
```

Blank()

Description

Blanks the initial velocity generation

Arguments

No arguments

Returns

No return value

Example

To blank initial velocity generation ivg:

```
ivg.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the initial velocity generations in the model.

Arguments

- **Model** (Model)

Model that all initial velocity generations will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the initial velocity generations in model m:

```
VelocityGeneration.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged initial velocity generations in the model.

Arguments

- **Model** (Model)

Model that all the flagged initial velocity generations will be blanked in

- **flag** (Flag)

Flag set on the initial velocity generations that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the initial velocity generations in model m flagged with f:

```
VelocityGeneration.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the initial velocity generation is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if initial velocity generation ivg is blanked:

```
if (ivg.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse initial velocity generation ivg:

```
ivg.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the initial velocity generation.

Arguments

- **flag** (Flag)

Flag to clear on the initial velocity generation

Returns

No return value

Example

To clear flag f for initial velocity generation ivg:

```
ivg.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the initial velocity generation. The target include of the copied initial velocity generation can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

VelocityGeneration object

Return type

VelocityGeneration

Example

To copy initial velocity generation ivg into initial velocity generation z:

```
var z = ivg.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a initial velocity generation

Arguments

- **Model** (Model)

Model that the initial velocity generation will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

VelocityGeneration object (or null if not made).

Return type

VelocityGeneration

Example

To start creating a initial velocity generation in model m:

```
var ivg = VelocityGeneration.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a initial velocity generation.

Arguments

- **Comment** (Comment)

Comment that will be detached from the initial velocity generation

Returns

No return value

Example

To detach comment c from the initial velocity generation ivg:

```
ivg.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit initial velocity generation ivg:

```
ivg.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for initial velocity generation with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The initial velocity generation object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for initial velocity generation ivg:

```
ivg.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for initial velocity generation ivg:

```
ivg.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for initial velocity generation. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for initial velocity generation ivg:

```
ivg.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first initial velocity generation in the model.

Arguments

- **Model** (Model)

Model to get first initial velocity generation in

Returns

VelocityGeneration object (or null if there are no initial velocity generations in the model).

Return type

VelocityGeneration

Example

To get the first initial velocity generation in model m:

```
var ivg = VelocityGeneration.First(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the initial velocity generations in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all initial velocity generations will be flagged in

- **flag** (Flag)

Flag to set on the initial velocity generations

Returns

No return value

Example

To flag all of the initial velocity generations with flag f in model m:

```
VelocityGeneration.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the initial velocity generation is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the initial velocity generation

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if initial velocity generation ivg has flag f set on it:

```
if (ivg.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each initial velocity generation in the model.
Note that ForEach has been designed to make looping over initial velocity generations as fast as possible and so has some limitations.
Firstly, a single temporary VelocityGeneration object is created and on each function call it is updated with the current initial velocity generation data. This means that you should not try to store the VelocityGeneration object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new initial velocity generations inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all initial velocity generations are in

- **func** (function)

Function to call for each initial velocity generation

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the initial velocity generations in model m:

```
VelocityGeneration.ForEach(m, test);
function test(ivg)
{
// ivg is VelocityGeneration object
}
```

To call function test for all of the initial velocity generations in model m with optional object:

```
var data = { x:0, y:0 };
VelocityGeneration.ForEach(m, test, data);
function test(ivg, extra)
{
// ivg is VelocityGeneration object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of VelocityGeneration objects or properties for all of the initial velocity generations in a model in PRIMER. If the optional property argument is not given then an array of VelocityGeneration objects is returned. If the property argument is given, that property value for each initial velocity generation is returned in the array instead of a VelocityGeneration object

Arguments

- **Model** (Model)

Model to get initial velocity generations from

- **property (optional)** (string)

Name for property to get for all initial velocity generations in the model

Returns

Array of VelocityGeneration objects or properties

Return type

Array

Example

To make an array of VelocityGeneration objects for all of the initial velocity generations in model m:

```
var a = VelocityGeneration.GetAll(m);
```

To return an array containing the value of property ’foo’ (for example ’x’ for a node) for each initial velocity generation in model m:

```
var a = VelocityGeneration.GetAll(m, ’foo’);
```

GetComments()

Description

Extracts the comments associated to a initial velocity generation.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the initial velocity generation ivg:

```
var comm_array = ivg.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of VelocityGeneration objects for all of the flagged initial velocity generations in a model in PRIMER If the optional property argument is not given then an array of VelocityGeneration objects is returned. If the property argument is given, then that property value for each initial velocity generation is returned in the array instead of a VelocityGeneration object

Arguments

- **Model** (Model)

Model to get initial velocity generations from

- **flag** (Flag)

Flag set on the initial velocity generations that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged initial velocity generations in the model

Returns

Array of VelocityGeneration objects or properties

Return type

Array

Example

To make an array of VelocityGeneration objects for all of the initial velocity generations in model m flagged with f:

```
var ivg = VelocityGeneration.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the initial velocity generations in model m flagged with f:

```
var a = VelocityGeneration.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the VelocityGeneration object for a initial velocity generation ID.

Arguments

- **Model** (Model)

Model to find the initial velocity generation in

- **number** (integer)

number of the initial velocity generation you want the VelocityGeneration object for

Returns

VelocityGeneration object (or null if initial velocity generation does not exist).

Return type

VelocityGeneration

Example

To get the VelocityGeneration object for initial velocity generation 100 in model m

```
var ivg = VelocityGeneration.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a VelocityGeneration property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the VelocityGeneration.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

initial velocity generation property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if VelocityGeneration property ivg.example is a parameter:

```
Options.property_parameter_names = true;
if (ivg.GetParameter(ivg.example) ) do_something...
Options.property_parameter_names = false;
```

To check if VelocityGeneration property ivg.example is a parameter by using the GetParameter method:

```
if (ivg.ViewParameters().GetParameter(ivg.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this initial velocity (*INITIAL_VELOCITY_GENERATION). **Note that a carriage return is not added**. See also VelocityGeneration.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for velocity i:

```
var key = i.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the initial velocity_generation. **Note that a carriage return is not added**. See also VelocityGeneration.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for velocity i:

```
var cards = i.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last initial velocity generation in the model.

Arguments

- **Model** (Model)

Model to get last initial velocity generation in

Returns

VelocityGeneration object (or null if there are no initial velocity generations in the model).

Return type

VelocityGeneration

Example

To get the last initial velocity generation in model m:

```
var ivg = VelocityGeneration.Last(m);
```

Next()

Description

Returns the next initial velocity generation in the model.

Arguments

No arguments

Returns

VelocityGeneration object (or null if there are no more initial velocity generations in the model).

Return type

VelocityGeneration

Example

To get the initial velocity generation in model m after initial velocity generation ivg:

```
var ivg = ivg.Next();
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a initial velocity generation.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only initial velocity generations from that model can be picked. If the argument is a Flag then only initial velocity generations that are flagged with *limit* can be selected. If omitted, or null, any initial velocity generations from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

VelocityGeneration object (or null if not picked)

Return type

VelocityGeneration

Example

To pick a initial velocity generation from model m giving the prompt 'Pick initial velocity generation from screen':

```
var ivg = VelocityGeneration.Pick('Pick initial velocity generation from
screen', m);
```

Previous()

Description

Returns the previous initial velocity generation in the model.

Arguments

No arguments

Returns

VelocityGeneration object (or null if there are no more initial velocity generations in the model).

Return type

VelocityGeneration

Example

To get the initial velocity generation in model m before initial velocity generation ivg:

```
var ivg = ivg.Previous();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select initial velocity generations using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting initial velocity generations

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only initial velocity generations from that model can be selected. If the argument is a Flag then only initial velocity generations that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any initial velocity generations can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of initial velocity generations selected or null if menu cancelled

Return type

Number

Example

To select initial velocity generations from model m, flagging those selected with flag f, giving the prompt 'Select initial velocity generations':

```
VelocityGeneration.Select(f, 'Select initial velocity generations', m);
```

To select initial velocity generations, flagging those selected with flag f but limiting selection to initial velocity generations flagged with flag l, giving the prompt 'Select initial velocity generations':

```
VelocityGeneration.Select(f, 'Select initial velocity generations', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the initial velocity generation.

Arguments

- **flag** (Flag)

Flag to set on the initial velocity generation

Returns

No return value

Example

To set flag f for initial velocity generation ivg:

```
ivg.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the initial velocity generation. The initial velocity generation will be sketched until you either call VelocityGeneration.Unsketch(), VelocityGeneration.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the initial velocity generation is sketched. If omitted redraw is true. If you want to sketch several initial velocity generations and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch initial velocity generation ivg:

```
ivg.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged initial velocity generations in the model. The initial velocity generations will be sketched until you either call VelocityGeneration.Unsketch(), VelocityGeneration.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged initial velocity generations will be sketched in

- **flag** (Flag)

Flag set on the initial velocity generations that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the initial velocity generations are sketched. If omitted redraw is true. If you want to sketch flagged initial velocity generations several times and only redraw after the last one then use false for

redraw and call View.Redraw().

Returns

No return value

Example

To sketch all initial velocity generations flagged with flag in model m:

```
VelocityGeneration.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of initial velocity generations in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing initial velocity generations should be counted. If false or omitted referenced but undefined initial velocity generations will also be included in the total.

Returns

number of initial velocity generations

Return type

Number

Example

To get the total number of initial velocity generations in model m:

```
var total = VelocityGeneration.Total(m);
```

Unblank()

Description

Unblanks the initial velocity generation

Arguments

No arguments

Returns

No return value

Example

To unblank initial velocity generation ivg:

```
ivg.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the initial velocity generations in the model.

Arguments

- **Model** (Model)

Model that all initial velocity generations will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the initial velocity generations in model m:

```
VelocityGeneration.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged initial velocity generations in the model.

Arguments

- **Model** (Model)

Model that the flagged initial velocity generations will be unblanked in

- **flag** (Flag)

Flag set on the initial velocity generations that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the initial velocity generations in model m flagged with f:

```
VelocityGeneration.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the initial velocity generations in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all initial velocity generations will be unset in

- **flag** (Flag)

Flag to unset on the initial velocity generations

Returns

No return value

Example

To unset the flag f on all the initial velocity generations in model m:

```
VelocityGeneration.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the initial velocity generation.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the initial velocity generation is unsketched. If omitted redraw is true. If you want to unsketch several initial velocity generations and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch initial velocity generation ivg:

```
ivg.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all initial velocity generations.

Arguments

- **Model** (Model)

Model that all initial velocity generations will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the initial velocity generations are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all initial velocity generations in model m:

```
VelocityGeneration.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged initial velocity generations in the model.

Arguments

- **Model** (Model)

Model that all initial velocity generations will be unsketched in

- **flag** (Flag)

Flag set on the initial velocity generations that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the initial velocity generations are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all initial velocity generations flagged with flag in model m:

```
VelocityGeneration.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

VelocityGeneration object.

Return type

VelocityGeneration

Example

To check if VelocityGeneration property ivg.example is a parameter by using the VelocityGeneration.GetParameter() method:

```
if (ivg.ViewParameters().GetParameter(ivg.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for initial velocity generation with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The initial velocity generation object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for initial velocity generation ivg:

```
ivg.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for initial velocity generation ivg:

```
ivg.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for initial velocity generation. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for initial velocity generation ivg:

```
ivg.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this initial velocity generation.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for initial velocity generation ivg:

```
var xrefs = ivg.Xrefs();
```

---

toString()

Description

Creates a string containing the initial velocity data in keyword format. Note that this contains the keyword header and the keyword cards. See also VelocityGeneration.Keyword() and VelocityGeneration.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for velocity i in keyword format

```
var s = i.toString();
```

IntegrationBeam (IntB) class

The IntegrationBeam class gives you access to integration beam cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetIntegrationPoint(index*[integer]*)
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- SetIntegrationPoint(index*[integer]*, s*[real]*, t*[real]*, wf*[real]*, pid(optional)*[integer]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

IntegrationBeam properties

Name	Type	Description
d1	real	Cross-section dimension.
d2	real	Cross-section dimension.

d3	real	Cross-section dimension.
d4	real	Cross-section dimension.
d5	real	Cross-section dimension.
d6	real	Cross-section dimension.
exists (read only)	logical	true if integration beam exists, false if referred to but not defined.
icst	integer	Standard cross section type. If icst is non-zero, nip should be zero and vice-versa.
include	integer	The Include file number that the integration beam is in.
irid	integer	Integration rule id.
k	integer	Integration refinement parameter for standard cross section types.
model (read only)	integer	The Model number that the integration beam is in.
nip	integer	Number of integration points. If nip is non-zero, icst should be zero and vice-versa.
pid	Part	Optional part ID if different from the PID specified on the element card.
ra	real	Relative area of cross section.
s	real	Normalized s coordinate of integration point.
sref	real	Location of reference surface normal to s, for the Hughes-Liu beam only.
t	real	Normalized t coordinate of integration point.
tref	real	Location of reference surface normal to t, for the Hughes-Liu beam only.
wf	real	Weighting factor (area associated with integration point divided by actual cross sectional area).

Detailed Description

The IntegrationBeam class allows you to create, modify, edit and manipulate integration beam cards. See the documentation below for more details.

For convenience "IntB" can also be used as the class name instead of "IntegrationBeam".

Constructor

new IntegrationBeam(Model*[Model]*, irid*[integer]*, nip (optional)*[integer]*, ra (optional)*[real]*, icst (optional)*[integer]*, k (optional)*[integer]*)

Description

Create a new IntegrationBeam object.

Arguments

- **Model** (Model)

Model that intb will be created in.

- **irid** (integer)

Integration_Beam ID.

- **nip (optional)** (integer)

Number of integration points. If omitted nip will be 0. If nip is non-zero, icst should be zero and vice-versa.

- **ra (optional)** (real)

Relative area of cross section. If omitted ra will be 0.

- **icst (optional)** (integer)

Standard cross section type. If omitted icst will be 0. If icst is non-zero, nip should be zero and vice-versa.

- **k (optional)** (integer)

Integration refinement parameter for standard cross section types. If omitted k will be 0.

Returns

IntegrationBeam object

Return type

IntegrationBeam

Example

To create a new intgb 1000 in model m with the following specifiction: irid, nip, ra, icst, k are 1000, 2, 0.1, 3, 5 respectively

```
var w = new IntegrationBeam(m, 1000, 2, 0.1, 3, 5);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a integration beam.

Arguments

- **Comment** (Comment)

Comment that will be attached to the integration beam

Returns

No return value

Example

To associate comment c to the integration beam ib:

```
ib.AssociateComment(c);
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse integration beam ib:

```
ib.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the integration beam.

Arguments

- **flag** (Flag)

Flag to clear on the integration beam

Returns

No return value

Example

To clear flag f for integration beam ib:

```
ib.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the integration beam. The target include of the copied integration beam can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

IntegrationBeam object

Return type

IntegrationBeam

Example

To copy integration beam ib into integration beam z:

```
var z = ib.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a integration beam

Arguments

- **Model** (Model)

Model that the integration beam will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

IntegrationBeam object (or null if not made).

Return type

IntegrationBeam

Example

To start creating a integration beam in model m:

```
var ib = IntegrationBeam.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a integration beam.

Arguments

- **Comment** (Comment)

Comment that will be detached from the integration beam

Returns

No return value

Example

To detach comment c from the integration beam ib:

```
ib.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit integration beam ib:

```
ib.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for integration beam with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The integration beam object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for integration beam ib:

```
ib.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for integration beam ib:

```
ib.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for integration beam. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for integration beam ib:

```
ib.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first integration beam in the model.

Arguments

- **Model** (Model)

Model to get first integration beam in

Returns

IntegrationBeam object (or null if there are no integration beams in the model).

Return type

IntegrationBeam

Example

To get the first integration beam in model m:

```
var ib = IntegrationBeam.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free integration beam label in the model. Also see IntegrationBeam.LastFreeLabel(), IntegrationBeam.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free integration beam label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

IntegrationBeam label.

Return type

Number

Example

To get the first free integration beam label in model m:

```
var label = IntegrationBeam.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the integration beams in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all integration beams will be flagged in

- **flag** (Flag)

Flag to set on the integration beams

Returns

No return value

Example

To flag all of the integration beams with flag f in model m:

```
IntegrationBeam.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the integration beam is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the integration beam

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if integration beam ib has flag f set on it:

```
if (ib.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each integration beam in the model.
Note that ForEach has been designed to make looping over integration beams as fast as possible and so has some limitations.
Firstly, a single temporary IntegrationBeam object is created and on each function call it is updated with the current integration beam data. This means that you should not try to store the IntegrationBeam object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new integration beams inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all integration beams are in

- **func** (function)

Function to call for each integration beam

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the integration beams in model m:

```
IntegrationBeam.ForEach(m, test);
function test(ib)
{
// ib is IntegrationBeam object
}
```

To call function test for all of the integration beams in model m with optional object:

```
var data = { x:0, y:0 };
IntegrationBeam.ForEach(m, test, data);
function test(ib, extra)
{
// ib is IntegrationBeam object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of IntegrationBeam objects or properties for all of the integration beams in a model in PRIMER. If the optional property argument is not given then an array of IntegrationBeam objects is returned. If the property argument is given, that property value for each integration beam is returned in the array instead of a IntegrationBeam object

Arguments

- **Model** (Model)

Model to get integration beams from

- **property (optional)** (string)

Name for property to get for all integration beams in the model

Returns

Array of IntegrationBeam objects or properties

Return type

Array

Example

To make an array of IntegrationBeam objects for all of the integration beams in model m:

```
var a = IntegrationBeam.GetAll(m);
```

To return an array containing the value of property ’foo’ (for example ’x’ for a node) for each integration beam in model m:

```
var a = IntegrationBeam.GetAll(m, ’foo’);
```

GetComments()

Description

Extracts the comments associated to a integration beam.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the integration beam ib:

```
var comm_array = ib.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of IntegrationBeam objects for all of the flagged integration beams in a model in PRIMER If the optional property argument is not given then an array of IntegrationBeam objects is returned. If the property argument is given, then that property value for each integration beam is returned in the array instead of a IntegrationBeam object

Arguments

- **Model** (Model)

Model to get integration beams from

- **flag** (Flag)

Flag set on the integration beams that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged integration beams in the model

Returns

Array of IntegrationBeam objects or properties

Return type

Array

Example

To make an array of IntegrationBeam objects for all of the integration beams in model m flagged with f:

```
var ib = IntegrationBeam.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the integration beams in model m flagged with f:

```
var a = IntegrationBeam.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the IntegrationBeam object for a integration beam ID.

Arguments

- **Model** (Model)

Model to find the integration beam in

- **number** (integer)

number of the integration beam you want the IntegrationBeam object for

Returns

IntegrationBeam object (or null if integration beam does not exist).

Return type

IntegrationBeam

Example

To get the IntegrationBeam object for integration beam 100 in model m

```
var ib = IntegrationBeam.GetFromID(m, 100);
```

GetIntegrationPoint(index*[integer]*)

Description

Returns the data for an integration point in *INTEGRATION_BEAM.**Note data is only available when NIP>0**.

Arguments

- **index** (integer)

Index you want the integration point data for. **Note that indices start at 0**.

Returns

An array containing the integration point data.

Return type

Array

Example

To get the data for the 3rd integration point for integration beam ib:

```
var data = ib.GetIntegrationPoint(2);
```

GetParameter(prop*[string]*)

Description

Checks if a IntegrationBeam property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the IntegrationBeam.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

integration beam property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if IntegrationBeam property ib.example is a parameter:

```
Options.property_parameter_names = true;
if (ib.GetParameter(ib.example) ) do_something...
Options.property_parameter_names = false;
```

To check if IntegrationBeam property ib.example is a parameter by using the GetParameter method:

```
if (ib.ViewParameters().GetParameter(ib.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this intb (*INTEGRATION_BEAM). **Note that a carriage return is not added**. See also IntegrationBeam.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for intb n:

```
var key = n.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the intb. **Note that a carriage return is not added**. See also IntegrationBeam.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for intb n:

```
var cards = n.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last integration beam in the model.

Arguments

- **Model** (Model)

Model to get last integration beam in

Returns

IntegrationBeam object (or null if there are no integration beams in the model).

Return type

IntegrationBeam

Example

To get the last integration beam in model m:

```
var ib = IntegrationBeam.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free integration beam label in the model. Also see IntegrationBeam.FirstFreeLabel(), IntegrationBeam.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free integration beam label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

IntegrationBeam label.

Return type

Number

Example

To get the last free integration beam label in model m:

```
var label = IntegrationBeam.LastFreeLabel(m);
```

Next()

Description

Returns the next integration beam in the model.

Arguments

No arguments

Returns

IntegrationBeam object (or null if there are no more integration beams in the model).

Return type

IntegrationBeam

Example

To get the integration beam in model m after integration beam ib:

```
var ib = ib.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) integration beam label in the model. Also see IntegrationBeam.FirstFreeLabel(), IntegrationBeam.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free integration beam label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

IntegrationBeam label.

Return type

Number

Example

To get the next free integration beam label in model m:

```
var label = IntegrationBeam.NextFreeLabel(m);
```

Previous()

Description

Returns the previous integration beam in the model.

Arguments

No arguments

Returns

IntegrationBeam object (or null if there are no more integration beams in the model).

Return type

IntegrationBeam

Example

To get the integration beam in model m before integration beam ib:

```
var ib = ib.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the integration beams in the model.

Arguments

- **Model** (Model)

Model that all integration beams will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the integration beams in model m, from 1000000:

```
IntegrationBeam.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged integration beams in the model.

Arguments

- **Model** (Model)

Model that all the flagged integration beams will be renumbered in

- **flag** (Flag)

Flag set on the integration beams that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the integration beams in model m flagged with f, from 1000000:

```
IntegrationBeam.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select integration beams using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting integration beams

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only integration beams from that model can be selected. If the argument is a Flag then only integration beams that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any integration beams can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of integration beams selected or null if menu cancelled

Return type

Number

Example

To select integration beams from model m, flagging those selected with flag f, giving the prompt 'Select integration beams':

```
IntegrationBeam.Select(f, 'Select integration beams', m);
```

To select integration beams, flagging those selected with flag f but limiting selection to integration beams flagged with flag l, giving the prompt 'Select integration beams':

```
IntegrationBeam.Select(f, 'Select integration beams', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the integration beam.

Arguments

- **flag** (Flag)

Flag to set on the integration beam

Returns

No return value

Example

To set flag f for integration beam ib:

```
ib.SetFlag(f);
```

SetIntegrationPoint(index*[integer]*, s*[real]*, t*[real]*, wf*[real]*, pid(optional)*[integer]*)

Description

Sets the integration point data for an *INTEGRATION_BEAM.

Arguments

- **index** (integer)

Index you want to set the integration point data for. **Note that indices start at 0**.

- **s** (real)

s coordinate of integration point in range -1 to 1.

- **t** (real)

s coordinate of integration point in range -1 to 1.

- **wf** (real)

Weighting factor, area associated with the integration point divided by actual beam cross sectional area.

- **pid(optional)** (integer)

Optional part ID if different from the PID specified on the element card.

Returns

No return value.

Example

To set the 4th integration point for *INTEGRATION_BEAM ib to the following specifiction: s, t, wf, pid are 0.1, 0.2, 0.3, 1 respectively

```
ib.SetIntegrationPoint(3, 0.1, 0.2, 0.3, 1);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of integration beams in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing integration beams should be counted. If false or omitted referenced but undefined integration beams will also be included in the total.

Returns

number of integration beams

Return type

Number

Example

To get the total number of integration beams in model m:

```
var total = IntegrationBeam.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the integration beams in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all integration beams will be unset in

- **flag** (Flag)

Flag to unset on the integration beams

Returns

No return value

Example

To unset the flag f on all the integration beams in model m:

```
IntegrationBeam.UnflagAll(m, f);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

IntegrationBeam object.

Return type

IntegrationBeam

Example

To check if IntegrationBeam property ib.example is a parameter by using the IntegrationBeam.GetParameter() method:

```
if (ib.ViewParameters().GetParameter(ib.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for integration beam with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The integration beam object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for integration beam ib:

```
ib.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for integration beam ib:

```
ib.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for integration beam. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for integration beam ib:

```
ib.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this integration beam.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for integration beam ib:

```
var xrefs = ib.Xrefs();
```

---

toString()

Description

Creates a string containing the intb data in keyword format. Note that this contains the keyword header and the keyword cards. See also IntegrationBeam.Keyword() and IntegrationBeam.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for intb n in keyword format

```
var s = n.toString();
```

IntegrationShell (IntS) class

The IntegrationShell class gives you access to integration shell cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetIntegrationPoint(index*[integer]*)
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- SetIntegrationPoint(index*[integer]*, s*[real]*, wf*[real]*, pid(optional)*[integer]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

IntegrationShell properties

Name	Type	Description
esop	integer	Equal spacing of integration points option.
exists (read only)	logical	true if integration shell exists, false if referred to but not defined.

failopt	integer	Treatment of failure when mixing different constitutive types.
include	integer	The Include file number that the integration shell is in.
irid	integer	Integration rule id.
model (read only)	integer	The Model number that the integration shell is in.
nip	integer	Number of integration points.
pid	Part	Optional part ID if different from the PID specified on the element card.
s	real	Coordinate of integration point in range -1 to 1.
wf	real	Weighting factor (thickness associated with integration point divided by actual shell thickness).

Detailed Description

The IntegrationShell class allows you to create, modify, edit and manipulate integration shell cards. See the documentation below for more details.

For convenience "IntS" can also be used as the class name instead of "IntegrationShell".

Constructor

new IntegrationShell(Model*[Model]*, irid*[integer]*, nip*[integer]*, esop (optional)*[integer]*, failopt (optional)*[integer]*)

Description

Create a new IntegrationShell object.

Arguments

- **Model** (Model)

Model that ints will be created in.

- **irid** (integer)

Integration_Beam ID.

- **nip** (integer)

Number of integration points.

- **esop (optional)** (integer)

Equal spacing of integration points option. If omitted esop will be 0.

- **failopt (optional)** (integer)

Treatment of failure when mixing different constitutive types. If omitted failopt will be 0.

Returns

IntegrationShell object

Return type

IntegrationShell

Example

To create a new ints 1000 in model m with the following specifiction: irid, nip, esop, failopt are 1000, 2, 0, 1 respectively

```
var w = new IntegrationBeam(m, 1000, 2, 0, 1);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a integration shell.

Arguments

- **Comment** (Comment)

Comment that will be attached to the integration shell

Returns

No return value

Example

To associate comment c to the integration shell is:

```
is.AssociateComment(c);
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse integration shell is:

```
is.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the integration shell.

Arguments

- **flag** (Flag)

Flag to clear on the integration shell

Returns

No return value

Example

To clear flag f for integration shell is:

```
is.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the integration shell. The target include of the copied integration shell can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

IntegrationShell object

Return type

IntegrationShell

Example

To copy integration shell is into integration shell z:

```
var z = is.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a integration shell

Arguments

- **Model** (Model)

Model that the integration shell will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

IntegrationShell object (or null if not made).

Return type

IntegrationShell

Example

To start creating a integration shell in model m:

```
var is = IntegrationShell.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a integration shell.

Arguments

- **Comment** (Comment)

Comment that will be detached from the integration shell

Returns

No return value

Example

To detach comment c from the integration shell is:

```
is.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit integration shell is:

```
is.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for integration shell with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description

autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The integration shell object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for integration shell is:

```
is.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for integration shell is:

```
is.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for integration shell. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for integration shell is:

```
is.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first integration shell in the model.

Arguments

- **Model** (Model)

Model to get first integration shell in

Returns

IntegrationShell object (or null if there are no integration shells in the model).

Return type

IntegrationShell

Example

To get the first integration shell in model m:

```
var is = IntegrationShell.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free integration shell label in the model. Also see IntegrationShell.LastFreeLabel(), IntegrationShell.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free integration shell label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

IntegrationShell label.

Return type

Number

Example

To get the first free integration shell label in model m:

```
var label = IntegrationShell.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the integration shells in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all integration shells will be flagged in

- **flag** (Flag)

Flag to set on the integration shells

Returns

No return value

Example

To flag all of the integration shells with flag f in model m:

```
IntegrationShell.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the integration shell is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the integration shell

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if integration shell is has flag f set on it:

```
if (is.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each integration shell in the model.
Note that ForEach has been designed to make looping over integration shells as fast as possible and so has some limitations.
Firstly, a single temporary IntegrationShell object is created and on each function call it is updated with the current integration shell data. This means that you should not try to store the IntegrationShell object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new integration shells inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all integration shells are in

- **func** (function)

Function to call for each integration shell

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the integration shells in model m:

```
IntegrationShell.ForEach(m, test);
function test(is)
{
// is is IntegrationShell object
}
```

To call function test for all of the integration shells in model m with optional object:

```
var data = { x:0, y:0 };
IntegrationShell.ForEach(m, test, data);
function test(is, extra)
{
// is is IntegrationShell object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of IntegrationShell objects or properties for all of the integration shells in a model in PRIMER. If the optional property argument is not given then an array of IntegrationShell objects is returned. If the property argument is given, that property value for each integration shell is returned in the array instead of a IntegrationShell object

Arguments

- **Model** (Model)

Model to get integration shells from

- **property (optional)** (string)

Name for property to get for all integration shells in the model

Returns

Array of IntegrationShell objects or properties

Return type

Array

Example

To make an array of IntegrationShell objects for all of the integration shells in model m:

```
var a = IntegrationShell.GetAll(m);
```

To return an array containing the value of property ’foo’ (for example ’x’ for a node) for each integration shell in model m:

```
var a = IntegrationShell.GetAll(m, ’foo’);
```

GetComments()

Description

Extracts the comments associated to a integration shell.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the integration shell is:

```
var comm_array = is.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of IntegrationShell objects for all of the flagged integration shells in a model in PRIMER If the optional property argument is not given then an array of IntegrationShell objects is returned. If the property argument is given, then that property value for each integration shell is returned in the array instead of a IntegrationShell object

Arguments

- **Model** (Model)

Model to get integration shells from

- **flag** (Flag)

Flag set on the integration shells that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged integration shells in the model

Returns

Array of IntegrationShell objects or properties

Return type

Array

Example

To make an array of IntegrationShell objects for all of the integration shells in model m flagged with f:

```
var is = IntegrationShell.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the integration shells in model m flagged with f:

```
var a = IntegrationShell.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the IntegrationShell object for a integration shell ID.

Arguments

- **Model** (Model)

Model to find the integration shell in

- **number** (integer)

number of the integration shell you want the IntegrationShell object for

Returns

IntegrationShell object (or null if integration shell does not exist).

Return type

IntegrationShell

Example

To get the IntegrationShell object for integration shell 100 in model m

```
var is = IntegrationShell.GetFromID(m, 100);
```

GetIntegrationPoint(index*[integer]*)

Description

Returns the data for an integration point in *INTEGRATION_SHELL.**Note data is only available when NIP>0 and ESOP=0**.

Arguments

- **index** (integer)

Index you want the integration point data for. **Note that indices start at 0**.

Returns

An array containing the integration point data.

Return type

Array

Example

To get the data for the 3rd integration point for integration shell:

```
var data = is.GetIntegrationPoint(2);
```

GetParameter(prop*[string]*)

Description

Checks if a IntegrationShell property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the IntegrationShell.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

integration shell property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if IntegrationShell property is.example is a parameter:

```
Options.property_parameter_names = true;
if (is.GetParameter(is.example) ) do_something...
Options.property_parameter_names = false;
```

To check if IntegrationShell property is.example is a parameter by using the GetParameter method:

```
if (is.ViewParameters().GetParameter(is.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this ints (*INTEGRATION_SHELL). **Note that a carriage return is not added**. See also IntegrationShell.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for ints n:

```
var key = n.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the ints. **Note that a carriage return is not added**. See also IntegrationShell.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for integration shell is:

```
var cards = is.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last integration shell in the model.

Arguments

- **Model** (Model)

Model to get last integration shell in

Returns

IntegrationShell object (or null if there are no integration shells in the model).

Return type

IntegrationShell

Example

To get the last integration shell in model m:

```
var is = IntegrationShell.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free integration shell label in the model. Also see IntegrationShell.FirstFreeLabel(), IntegrationShell.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free integration shell label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

IntegrationShell label.

Return type

Number

Example

To get the last free integration shell label in model m:

```
var label = IntegrationShell.LastFreeLabel(m);
```

Next()

Description

Returns the next integration shell in the model.

Arguments

No arguments

Returns

IntegrationShell object (or null if there are no more integration shells in the model).

Return type

IntegrationShell

Example

To get the integration shell in model m after integration shell is:

```
var is = is.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) integration shell label in the model. Also see IntegrationShell.FirstFreeLabel(), IntegrationShell.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free integration shell label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

IntegrationShell label.

Return type

Number

Example

To get the next free integration shell label in model m:

```
var label = IntegrationShell.NextFreeLabel(m);
```

Previous()

Description

Returns the previous integration shell in the model.

Arguments

No arguments

Returns

IntegrationShell object (or null if there are no more integration shells in the model).

Return type

IntegrationShell

Example

To get the integration shell in model m before integration shell is:

```
var is = is.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the integration shells in the model.

Arguments

- **Model** (Model)

Model that all integration shells will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the integration shells in model m, from 1000000:

```
IntegrationShell.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged integration shells in the model.

Arguments

- **Model** (Model)

Model that all the flagged integration shells will be renumbered in

- **flag** (Flag)

Flag set on the integration shells that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the integration shells in model m flagged with f, from 1000000:

```
IntegrationShell.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select integration shells using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting integration shells

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only integration shells from that model can be selected. If the argument is a Flag then only integration shells that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any integration shells can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of integration shells selected or null if menu cancelled

Return type

Number

Example

To select integration shells from model m, flagging those selected with flag f, giving the prompt 'Select integration shells':

```
IntegrationShell.Select(f, 'Select integration shells', m);
```

To select integration shells, flagging those selected with flag f but limiting selection to integration shells flagged with flag l, giving the prompt 'Select integration shells':

```
IntegrationShell.Select(f, 'Select integration shells', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the integration shell.

Arguments

- **flag** (Flag)

Flag to set on the integration shell

Returns

No return value

Example

To set flag f for integration shell is:

```
is.SetFlag(f);
```

SetIntegrationPoint(index*[integer]*, s*[real]*, wf*[real]*, pid(optional)*[integer]*)

Description

Sets the integration point data for an *INTEGRATION_SHELL.

Arguments

- **index** (integer)

Index you want to set the integration point data for. **Note that indices start at 0**.

- **s** (real)

Coordinate of integration point in range -1 to 1.

- **wf** (real)

Weighting factor, thickness associated with the integration point divided by actual shell thickness.

- **pid(optional)** (integer)

Optional part ID if different from the PID specified on the element card.

Returns

No return value.

Example

To set the 4th integration point for *INTEGRATION_SHELL is to the following specifiction: s, wf, pid are 0.1, 0.2, 1 respectively

```
is.SetIntegrationPoint(3, 0.1, 0.2, 1);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of integration shells in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing integration shells should be counted. If false or omitted referenced but undefined integration shells will also be included in the total.

Returns

number of integration shells

Return type

Number

Example

To get the total number of integration shells in model m:

```
var total = IntegrationShell.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the integration shells in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all integration shells will be unset in

- **flag** (Flag)

Flag to unset on the integration shells

Returns

No return value

Example

To unset the flag f on all the integration shells in model m:

```
IntegrationShell.UnflagAll(m, f);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

IntegrationShell object.

Return type

IntegrationShell

Example

To check if IntegrationShell property is.example is a parameter by using the IntegrationShell.GetParameter() method:

```
if (is.ViewParameters().GetParameter(is.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for integration shell with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The integration shell object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for integration shell is:

```
is.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for integration shell is:

```
is.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for integration shell. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for integration shell is:

```
is.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this integration shell.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for integration shell is:

```
var xrefs = is.Xrefs();
```

---

toString()

Description

Creates a string containing the ints data in keyword format. Note that this contains the keyword header and the keyword cards. See also IntegrationShell.Keyword() and IntegrationShell.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for integration shell is in keyword format

```
var s = is.toString();
```

InterfaceComponent class

The InterfaceComponent class gives you access to interface component cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

InterfaceComponent constants

Name	Description
InterfaceComponent.NODE	Node option
InterfaceComponent.SEGMENT	Segment option

InterfaceComponent properties

Name	Type	Description
cid	integer	Coordinate system ID.
exists (read only)	logical	true if interface component exists, false if referred to but not defined.
include	integer	The Include file number that the interface component is in.
model (read only)	integer	The Model number that the interface component is in.
nid	integer	Node ID.
nsid	integer	Element ID or element set ID. The ssid property is an alternative name for this.
option	constant	InterfaceComponent option. Can be InterfaceComponent.NODE, InterfaceComponent.SEGMENT,
ssid	integer	Element ID or element set ID. The nsid property is an alternative name for this.
title	string	InterfaceComponent title

Detailed Description

The InterfaceComponent class allows you to create, modify, edit and manipulate interface component cards. See the documentation below for more details.

Constructor

new InterfaceComponent(Model*[Model]*, type*[constant]*, snid/ssid*[integer]*, cid*[integer]*, nid*[integer]*, label (optional)*[integer]*, title (optional)*[string]*)

Description

Create a new InterfaceComponent object.

Arguments

- **Model** (Model)

Model that InterfaceComponent will be created in

- **type** (constant)

InterfaceComponent type. Can be InterfaceComponent.NODE, InterfaceComponent.SEGMENT,

- **snid/ssid** (integer)

Set node or set segment ID

- **cid** (integer)

Coordinate system ID

- **nid** (integer)

Node ID

- **label (optional)** (integer)

InterfaceComponent number

- **title (optional)** (string)

Title for this interface

Returns

InterfaceComponent object

Return type

InterfaceComponent

Example

To create a new Interface Component in model m with option: NODE, nsid: 100, cyd: 200, nid: 300, ID: 1, title: "MyInterfaceComponent"

```
var i_c = new InterfaceComponent(m, InterfaceComponent.NODE, 100, 200, 300, 1, MyInterfaceComponent);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a interface component.

Arguments

- **Comment** (Comment)

Comment that will be attached to the interface component

Returns

No return value

Example

To associate comment c to the interface component i_c:

```
i_c.AssociateComment(c);
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse interface component i_c:

```
i_c.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the interface component.

Arguments

- **flag** (Flag)

Flag to clear on the interface component

Returns

No return value

Example

To clear flag f for interface component i_c:

```
i_c.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the interface component. The target include of the copied interface component can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

InterfaceComponent object

Return type

InterfaceComponent

Example

To copy interface component i_c into interface component z:

```
var z = i_c.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a interface component

Arguments

- **Model** (Model)

Model that the interface component will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

InterfaceComponent object (or null if not made).

Return type

InterfaceComponent

Example

To start creating a interface component in model m:

```
var i_c = InterfaceComponent.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a interface component.

Arguments

- **Comment** (Comment)

Comment that will be detached from the interface component

Returns

No return value

Example

To detach comment c from the interface component i_c:

```
i_c.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit interface component i_c:

```
i_c.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for interface component with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The interface component object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for interface component i_c:

```
i_c.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for interface component i_c:

```
i_c.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for interface component. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for interface component i_c:

```
i_c.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first interface component in the model.

Arguments

- **Model** (Model)

Model to get first interface component in

Returns

InterfaceComponent object (or null if there are no interface components in the model).

Return type

InterfaceComponent

Example

To get the first interface component in model m:

```
var i_c = InterfaceComponent.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free interface component label in the model. Also see InterfaceComponent.LastFreeLabel(), InterfaceComponent.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free interface component label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

InterfaceComponent label.

Return type

Number

Example

To get the first free interface component label in model m:

```
var label = InterfaceComponent.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the interface components in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all interface components will be flagged in

- **flag** (Flag)

Flag to set on the interface components

Returns

No return value

Example

To flag all of the interface components with flag f in model m:

```
InterfaceComponent.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the interface component is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the interface component

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if interface component i_c has flag f set on it:

```
if (i_c.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each interface component in the model.
Note that ForEach has been designed to make looping over interface components as fast as possible and so has some limitations.
Firstly, a single temporary InterfaceComponent object is created and on each function call it is updated with the current interface component data. This means that you should not try to store the InterfaceComponent object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new interface components inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all interface components are in

- **func** (function)

Function to call for each interface component

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the interface components in model m:

```
InterfaceComponent.ForEach(m, test);
function test(i_c)
{
// i_c is InterfaceComponent object
}
```

To call function test for all of the interface components in model m with optional object:

```
var data = { x:0, y:0 };
InterfaceComponent.ForEach(m, test, data);
function test(i_c, extra)
{
// i_c is InterfaceComponent object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of InterfaceComponent objects or properties for all of the interface components in a model in PRIMER. If the optional property argument is not given then an array of InterfaceComponent objects is returned. If the property argument is given, that property value for each interface component is returned in the array instead of a InterfaceComponent object

Arguments

- **Model** (Model)

Model to get interface components from

- **property (optional)** (string)

Name for property to get for all interface components in the model

Returns

Array of InterfaceComponent objects or properties

Return type

Array

Example

To make an array of InterfaceComponent objects for all of the interface components in model m:

```
var a = InterfaceComponent.GetAll(m);
```

To return an array containing the value of property ’foo’ (for example ’x’ for a node) for each interface component in model m:

```
var a = InterfaceComponent.GetAll(m, ’foo’);
```

GetComments()

Description

Extracts the comments associated to a interface component.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the interface component i_c:

```
var comm_array = i_c.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of InterfaceComponent objects for all of the flagged interface components in a model in PRIMER If the optional property argument is not given then an array of InterfaceComponent objects is returned. If the property argument is given, then that property value for each interface component is returned in the array instead of a InterfaceComponent object

Arguments

- **Model** (Model)

Model to get interface components from

- **flag** (Flag)

Flag set on the interface components that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged interface components in the model

Returns

Array of InterfaceComponent objects or properties

Return type

Array

Example

To make an array of InterfaceComponent objects for all of the interface components in model m flagged with f:

```
var i_c = InterfaceComponent.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the interface components in model m flagged with f:

```
var a = InterfaceComponent.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the InterfaceComponent object for a interface component ID.

Arguments

- **Model** (Model)

Model to find the interface component in

- **number** (integer)

number of the interface component you want the InterfaceComponent object for

Returns

InterfaceComponent object (or null if interface component does not exist).

Return type

InterfaceComponent

Example

To get the InterfaceComponent object for interface component 100 in model m

```
var i_c = InterfaceComponent.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a InterfaceComponent property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the InterfaceComponent.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

interface component property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if InterfaceComponent property i_c.example is a parameter:

```
Options.property_parameter_names = true;
if (i_c.GetParameter(i_c.example) ) do_something...
Options.property_parameter_names = false;
```

To check if InterfaceComponent property i_c.example is a parameter by using the GetParameter method:

```
if (i_c.ViewParameters().GetParameter(i_c.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this InterfaceComponent (*INTERFACE_COMPONENT). **Note that a carriage return is not added**. See also InterfaceComponent.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for InterfaceComponent ed:

```
var key = ed.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the InterfaceComponent. **Note that a carriage return is not added**. See also InterfaceComponent.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for InterfaceComponent ed:

```
var cards = ed.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last interface component in the model.

Arguments

- **Model** (Model)

Model to get last interface component in

Returns

InterfaceComponent object (or null if there are no interface components in the model).

Return type

InterfaceComponent

Example

To get the last interface component in model m:

```
var i_c = InterfaceComponent.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free interface component label in the model. Also see InterfaceComponent.FirstFreeLabel(), InterfaceComponent.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free interface component label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

InterfaceComponent label.

Return type

Number

Example

To get the last free interface component label in model m:

```
var label = InterfaceComponent.LastFreeLabel(m);
```

Next()

Description

Returns the next interface component in the model.

Arguments

No arguments

Returns

InterfaceComponent object (or null if there are no more interface components in the model).

Return type

InterfaceComponent

Example

To get the interface component in model m after interface component i_c:

```
var i_c = i_c.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) interface component label in the model. Also see InterfaceComponent.FirstFreeLabel(), InterfaceComponent.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free interface component label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

InterfaceComponent label.

Return type

Number

Example

To get the next free interface component label in model m:

```
var label = InterfaceComponent.NextFreeLabel(m);
```

Previous()

Description

Returns the previous interface component in the model.

Arguments

No arguments

Returns

InterfaceComponent object (or null if there are no more interface components in the model).

Return type

InterfaceComponent

Example

To get the interface component in model m before interface component i_c:

```
var i_c = i_c.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the interface components in the model.

Arguments

- **Model** (Model)

Model that all interface components will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the interface components in model m, from 1000000:

```
InterfaceComponent.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged interface components in the model.

Arguments

- **Model** (Model)

Model that all the flagged interface components will be renumbered in

- **flag** (Flag)

Flag set on the interface components that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the interface components in model m flagged with f, from 1000000:

```
InterfaceComponent.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select interface components using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting interface components

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only interface components from that model can be selected. If the argument is a Flag then only interface components that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any interface components can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of interface components selected or null if menu cancelled

Return type

Number

Example

To select interface components from model m, flagging those selected with flag f, giving the prompt 'Select interface components':

```
InterfaceComponent.Select(f, 'Select interface components', m);
```

To select interface components, flagging those selected with flag f but limiting selection to interface components flagged with flag l, giving the prompt 'Select interface components':

```
InterfaceComponent.Select(f, 'Select interface components', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the interface component.

Arguments

- **flag** (Flag)

Flag to set on the interface component

Returns

No return value

Example

To set flag f for interface component i_c:

```
i_c.SetFlag(f);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of interface components in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing interface components should be counted. If false or omitted referenced but undefined interface components will also be included in the total.

Returns

number of interface components

Return type

Number

Example

To get the total number of interface components in model m:

```
var total = InterfaceComponent.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the interface components in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all interface components will be unset in

- **flag** (Flag)

Flag to unset on the interface components

Returns

No return value

Example

To unset the flag f on all the interface components in model m:

```
InterfaceComponent.UnflagAll(m, f);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

InterfaceComponent object.

Return type

InterfaceComponent

Example

To check if InterfaceComponent property i_c.example is a parameter by using the InterfaceComponent.GetParameter() method:

```
if (i_c.ViewParameters().GetParameter(i_c.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for interface component with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The interface component object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for interface component i_c:

```
i_c.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for interface component i_c:

```
i_c.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for interface component. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for interface component i_c:

```
i_c.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this interface component.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for interface component i_c:

```
var xrefs = i_c.Xrefs();
```

---

toString()

Description

Creates a string containing the InterfaceComponent data in keyword format. Note that this contains the keyword header and the keyword cards. See also InterfaceComponent.Keyword() and InterfaceComponent.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for InterfaceComponent ed in keyword format

```
var s = ed.toString();
```

InterfaceLinkingEdge class

The InterfaceLinkingEdge class gives you access to define Interface Linking Edge cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- First(Model*[Model]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

InterfaceLinkingEdge properties

Name	Type	Description
exists (read only)	logical	true if Interface Linking Edge exists, false if referred to but not defined.
ifid	integer	Interface ID.
include	integer	The Include file number that the Interface Linking Edge is in.
model (read only)	integer	The Model number that the Interface Linking Edge is in.
nsid	integer	Node set ID

Detailed Description

The InterfaceLinkingEdge class allows you to create, modify, edit and manipulate Interface Linking Edge cards. See the documentation below for more details.

Constructor

new InterfaceLinkingEdge(Model*[Model]*, nsid*[integer]*, ifid*[integer]*)

Description

Create a new InterfaceLinkingEdge object.

Arguments

- **Model** (Model)

Model that Interface Linking Edge will be created in

- **nsid** (integer)

Node set ID

- **ifid** (integer)

Interface ID

Returns

InterfaceLinkingEdge object

Return type

InterfaceLinkingEdge

Example

To create a new Interface Linking Edge in model m with NSID 900 and IFID 2

```
var b = new InterfaceLinkingEdge(m, 900, 2);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a Interface Linking Edge.

Arguments

- **Comment** (Comment)

Comment that will be attached to the Interface Linking Edge

Returns

No return value

Example

To associate comment c to the Interface Linking Edge I_LE:

```
I_LE.AssociateComment(c);
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the Interface Linking Edge.

Arguments

- **flag** (Flag)

Flag to clear on the Interface Linking Edge

Returns

No return value

Example

To clear flag f for Interface Linking Edge I_LE:

```
I_LE.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the Interface Linking Edge. The target include of the copied Interface Linking Edge can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

InterfaceLinkingEdge object

Return type

InterfaceLinkingEdge

Example

To copy Interface Linking Edge I_LE into Interface Linking Edge z:

```
var z = I_LE.Copy();
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a Interface Linking Edge.

Arguments

- **Comment** (Comment)

Comment that will be detached from the Interface Linking Edge

Returns

No return value

Example

To detach comment c from the Interface Linking Edge I_LE:

```
I_LE.DetachComment(c);
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for Interface Linking Edge with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The Interface Linking Edge object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for Interface Linking Edge I_LE:

```
I_LE.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for Interface Linking Edge I_LE:

```
I_LE.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for Interface Linking Edge. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for Interface Linking Edge I_LE:

```
I_LE.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first Interface Linking Edge in the model.

Arguments

- **Model** (Model)

Model to get first Interface Linking Edge in

Returns

InterfaceLinkingEdge object (or null if there are no Interface Linking Edges in the model).

Return type

InterfaceLinkingEdge

Example

To get the first Interface Linking Edge in model m:

```
var I_LE = InterfaceLinkingEdge.First(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the Interface Linking Edges in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all Interface Linking Edges will be flagged in

- **flag** (Flag)

Flag to set on the Interface Linking Edges

Returns

No return value

Example

To flag all of the Interface Linking Edges with flag f in model m:

```
InterfaceLinkingEdge.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the Interface Linking Edge is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the Interface Linking Edge

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if Interface Linking Edge I_LE has flag f set on it:

```
if (I_LE.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each Interface Linking Edge in the model.
Note that ForEach has been designed to make looping over Interface Linking Edges as fast as possible and so has some limitations.
Firstly, a single temporary InterfaceLinkingEdge object is created and on each function call it is updated with the current Interface Linking Edge data. This means that you should not try to store the InterfaceLinkingEdge object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new Interface Linking Edges inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all Interface Linking Edges are in

- **func** (function)

Function to call for each Interface Linking Edge

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the Interface Linking Edges in model m:

```
InterfaceLinkingEdge.ForEach(m, test);
function test(I_LE)
{
// I_LE is InterfaceLinkingEdge object
}
```

To call function test for all of the Interface Linking Edges in model m with optional object:

```
var data = { x:0, y:0 };
InterfaceLinkingEdge.ForEach(m, test, data);
function test(I_LE, extra)
{
// I_LE is InterfaceLinkingEdge object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of InterfaceLinkingEdge objects or properties for all of the Interface Linking Edges in a model in PRIMER. If the optional property argument is not given then an array of InterfaceLinkingEdge objects is returned. If the property argument is given, that property value for each Interface Linking Edge is returned in the array instead of a InterfaceLinkingEdge object

Arguments

- **Model** (Model)

Model to get Interface Linking Edges from

- **property (optional)** (string)

Name for property to get for all Interface Linking Edges in the model

Returns

Array of InterfaceLinkingEdge objects or properties

Return type

Array

Example

To make an array of InterfaceLinkingEdge objects for all of the Interface Linking Edges in model m:

```
var a = InterfaceLinkingEdge.GetAll(m);
```

To return an array containing the value of property ’foo’ (for example ’x’ for a node) for each Interface Linking Edge in model m:

```
var a = InterfaceLinkingEdge.GetAll(m, ’foo’);
```

GetComments()

Description

Extracts the comments associated to a Interface Linking Edge.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the Interface Linking Edge I_LE:

```
var comm_array = I_LE.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of InterfaceLinkingEdge objects for all of the flagged Interface Linking Edges in a model in PRIMER If the optional property argument is not given then an array of InterfaceLinkingEdge objects is returned. If the property argument is given, then that property value for each Interface Linking Edge is returned in the array instead of a InterfaceLinkingEdge object

Arguments

- **Model** (Model)

Model to get Interface Linking Edges from

- **flag** (Flag)

Flag set on the Interface Linking Edges that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged Interface Linking Edges in the model

Returns

Array of InterfaceLinkingEdge objects or properties

Return type

Array

Example

To make an array of InterfaceLinkingEdge objects for all of the Interface Linking Edges in model m flagged with f:

```
var I_LE = InterfaceLinkingEdge.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the Interface Linking Edges in model m flagged with f:

```
var a = InterfaceLinkingEdge.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the InterfaceLinkingEdge object for a Interface Linking Edge ID.

Arguments

- **Model** (Model)

Model to find the Interface Linking Edge in

- **number** (integer)

number of the Interface Linking Edge you want the InterfaceLinkingEdge object for

Returns

InterfaceLinkingEdge object (or null if Interface Linking Edge does not exist).

Return type

InterfaceLinkingEdge

Example

To get the InterfaceLinkingEdge object for Interface Linking Edge 100 in model m

```
var I_LE = InterfaceLinkingEdge.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a InterfaceLinkingEdge property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the InterfaceLinkingEdge.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

Interface Linking Edge property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if InterfaceLinkingEdge property I_LE.example is a parameter:

```
Options.property_parameter_names = true;
if (I_LE.GetParameter(I_LE.example) ) do_something...
Options.property_parameter_names = false;
```

To check if InterfaceLinkingEdge property I_LE.example is a parameter by using the GetParameter method:

```
if (I_LE.ViewParameters().GetParameter(I_LE.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this Interface Linking Edge (*INTERFACE_LINKING_EDGE). **Note that a carriage return is not added**. See also InterfaceLinkingEdge.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for Interface Linking Edge m:

```
var key = m.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the Interface Linking Edge. **Note that a carriage return is not added**. See also InterfaceLinkingEdge.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for Interface Linking Edge l:

```
var cards = l.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last Interface Linking Edge in the model.

Arguments

- **Model** (Model)

Model to get last Interface Linking Edge in

Returns

InterfaceLinkingEdge object (or null if there are no Interface Linking Edges in the model).

Return type

InterfaceLinkingEdge

Example

To get the last Interface Linking Edge in model m:

```
var I_LE = InterfaceLinkingEdge.Last(m);
```

Next()

Description

Returns the next Interface Linking Edge in the model.

Arguments

No arguments

Returns

InterfaceLinkingEdge object (or null if there are no more Interface Linking Edges in the model).

Return type

InterfaceLinkingEdge

Example

To get the Interface Linking Edge in model m after Interface Linking Edge I_LE:

```
var I_LE = I_LE.Next();
```

Previous()

Description

Returns the previous Interface Linking Edge in the model.

Arguments

No arguments

Returns

InterfaceLinkingEdge object (or null if there are no more Interface Linking Edges in the model).

Return type

InterfaceLinkingEdge

Example

To get the Interface Linking Edge in model m before Interface Linking Edge I_LE:

```
var I_LE = I_LE.Previous();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select Interface Linking Edges using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting Interface Linking Edges

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only Interface Linking Edges from that model can be selected. If the argument is a Flag then only Interface Linking Edges that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any Interface Linking Edges can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of Interface Linking Edges selected or null if menu cancelled

Return type

Number

Example

To select Interface Linking Edges from model m, flagging those selected with flag f, giving the prompt 'Select Interface Linking Edges':

```
InterfaceLinkingEdge.Select(f, 'Select Interface Linking Edges', m);
```

To select Interface Linking Edges, flagging those selected with flag f but limiting selection to Interface Linking Edges flagged with flag l, giving the prompt 'Select Interface Linking Edges':

```
InterfaceLinkingEdge.Select(f, 'Select Interface Linking Edges', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the Interface Linking Edge.

Arguments

- **flag** (Flag)

Flag to set on the Interface Linking Edge

Returns

No return value

Example

To set flag f for Interface Linking Edge I_LE:

```
I_LE.SetFlag(f);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of Interface Linking Edges in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing Interface Linking Edges should be counted. If false or omitted referenced but undefined Interface Linking Edges will also be included in the total.

Returns

number of Interface Linking Edges

Return type

Number

Example

To get the total number of Interface Linking Edges in model m:

```
var total = InterfaceLinkingEdge.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the Interface Linking Edges in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all Interface Linking Edges will be unset in

- **flag** (Flag)

Flag to unset on the Interface Linking Edges

Returns

No return value

Example

To unset the flag f on all the Interface Linking Edges in model m:

```
InterfaceLinkingEdge.UnflagAll(m, f);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

InterfaceLinkingEdge object.

Return type

InterfaceLinkingEdge

Example

To check if InterfaceLinkingEdge property I_LE.example is a parameter by using the InterfaceLinkingEdge.GetParameter() method:

```
if (I_LE.ViewParameters().GetParameter(I_LE.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for Interface Linking Edge with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The Interface Linking Edge object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for Interface Linking Edge I_LE:

```
I_LE.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for Interface Linking Edge I_LE:

```
I_LE.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for Interface Linking Edge. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for Interface Linking Edge I_LE:

```
I_LE.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this Interface Linking Edge.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for Interface Linking Edge I_LE:

```
var xrefs = I_LE.Xrefs();
```

---

toString()

Description

Creates a string containing the Interface Linking Edge data in keyword format. Note that this contains the keyword header and the keyword cards. See also InterfaceLinkingEdge.Keyword() and InterfaceLinkingEdge.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for Interface Linking Edge l in keyword format

```
var s = l.toString();
```

InterfaceSpringback class

The InterfaceSpringback class gives you access to interface springback cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetExcludeKeyword(idx*[integer]*)
- GetNodalPoint(npt*[integer]*)
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- RemoveExcludeKeyword(idx*[integer]*)
- RemoveNodalPoint(npt*[integer]*)
- SetExcludeKeyword(keystr*[string]*, index (optional)*[integer]*)
- SetFlag(flag*[Flag]*)
- SetNodalPoint(npt*[integer]*, nid*[integer]*, tc*[real]*, rc*[real]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

InterfaceSpringback constants

Constants for Types of Keyword

Name	Description
InterfaceSpringback.EXCLUDE	INTERFACE is *INTERFACE_SPRINGBACK_EXCLUDE.

InterfaceSpringback.LSDYNA	INTERFACE is *INTERFACE_SPRINGBACK_LSDYNA.
InterfaceSpringback.NASTRAN	INTERFACE is *INTERFACE_SPRINGBACK_NASTRAN.
InterfaceSpringback.NIKE3D	INTERFACE is *INTERFACE_SPRINGBACK_NIKE3D.
InterfaceSpringback.SEAMLESS	INTERFACE is *INTERFACE_SPRINGBACK_SEAMLESS.

InterfaceSpringback properties

Name	Type	Description
cflag	integer	Output contact state.
dtwrt	real	Time Interval.
exists (read only)	logical	true if interface springback exists, false if referred to but not defined.
fsplit	integer	Flag for splitting of the dynain file (0 - One file, 1 - Two files.). Used for OPTCARD field.
ftensr	integer	Flag for dumping tensor data from the element history variables into the dynain file (0/1).
ftype	integer	Filetype (0-3, 10-12).
hflag	integer	Output hourglass state.
include	integer	The Include file number that the interface springback is in.
intstrn	integer	Output of strains at all integration points of shell element is requested.
ivflg	integer	Initial volume output.
model (read only)	integer	The Model number that the interface springback is in.
ncyc	integer	Number of process cycles. Used for OPTCARD field.
ndflag	integer	Flag to dump nodes into dynain file.
nexclude	integer	gives the number of excluded keywords. Needed only for InterfaceSpringback.EXCLUDE.
nmwrt	integer	Maximum number of files.
nnodes	integer	gives the number of nodal points constrained for this keyword. (read_only)
nothickness	logical	true if _NOTHICKNESS (option2) is set. _NOTHICKNESS can be used only for InterfaceSpringback.LSDYNA or InterfaceSpringback.NASTRAN.
nshv	integer	Num additional Shell/Solid history variables number.
nthhsv	integer	Number of thermal history variables.
optcard	integer	(0) no optional cards (1) to have optional card 2 (2) to have optional cards 2, 3.1, 3.2.
psid	integer	Part set ID for springback.
sldo	integer	Output of solid element data as 0 - *ELEMENT_SOLID, 1- *ELEM_SOLID_ORTHO. Used for OPTCARD field.
type (read only)	integer	gives the type of InterfaceSpringback object.

Detailed Description

The InterfaceSpringback class allows you to create, modify, edit and manipulate interface springback cards. See the documentation below for more details.

Constructor

new InterfaceSpringback(Model*[Model]*, options *[object]*)

Description

Create a new InterfaceSpringback object.

Arguments

- **Model** (Model)

Model that interface springback will be created in

- **options** (object)

Options for creating the interface springback

Object has the following properties:

Name	Type	Description
cflag (optional)	integer	Output contact state.
dtwrt	real	Time Interval.
fsplit (optional)	integer	Flag for splitting of the dynain file (0 - One file, 1 - Two files.). Used only for optional card.
ftensr (optional)	integer	Flag for dumping tensor data from the element history variables into the dynain file (0/1).
ftype (optional)	integer	Filetype (0-3, 10-12).
hflag (optional)	integer	Output hourglass state.
intstrn (optional)	integer	Output of strains at all integration points of shell element is requested.
ivflg	integer	Initial volume output.
keylist (optional)	Array of strings	List of keywords to be excluded.(ONLY for EXCLUDE)
ncyc (optional)	integer	Number of process cycles. Used only for optional card.
ndflag (optional)	integer	Flag to dump nodes into dynain file.
nmwrt	integer	Maximum number of files.
nshv (optional)	integer	Num additional Shell/Solid history variables number.
nthhsv (optional)	integer	Number of thermal history variables.
optcard (optional)	integer	(0) no optional cards (1) to have optional card 2 (2) to have optional cards 2, 3.1, 3.2.
psid (optional)	integer	Part set ID for springback.(NOT for _EXCLUDE)
sldo (optional)	integer	Output of solid element data as 0 - *ELEMENT_SOLID, 1- *ELEM_SOLID_ORTHO. Used only for optional card.
type	constant	Specify the type of InterfaceSpringback (Can be InterfaceSpringback.NIKE3D or InterfaceSpringback.LSDYNA or InterfaceSpringback.NASTRAN or InterfaceSpringback.SEAMLESS)

Returns

InterfaceSpringback object

Return type

InterfaceSpringback

Example

To create a new interface springback in model m, type LSDYNA, part set id 100:

```
var i_s = new InterfaceSpringback(m, InterfaceSpringback.LSDYNA, 100);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a interface springback.

Arguments

- **Comment** (Comment)

Comment that will be attached to the interface springback

Returns

No return value

Example

To associate comment c to the interface springback i_s:

```
i_s.AssociateComment(c);
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse interface springback i_s:

```
i_s.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the interface springback.

Arguments

- **flag** (Flag)

Flag to clear on the interface springback

Returns

No return value

Example

To clear flag f for interface springback i_s:

```
i_s.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the interface springback. The target include of the copied interface springback can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

InterfaceSpringback object

Return type

InterfaceSpringback

Example

To copy interface springback i_s into interface springback z:

```
var z = i_s.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a interface springback

Arguments

- **Model** (Model)

Model that the interface springback will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

InterfaceSpringback object (or null if not made).

Return type

InterfaceSpringback

Example

To start creating a interface springback in model m:

```
var i_s = InterfaceSpringback.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a interface springback.

Arguments

- **Comment** (Comment)

Comment that will be detached from the interface springback

Returns

No return value

Example

To detach comment c from the interface springback i_s:

```
i_s.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit interface springback i_s:

```
i_s.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for interface springback with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The interface springback object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for interface springback i_s:

```
i_s.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for interface springback i_s:

```
i_s.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for interface springback. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for interface springback i_s:

```
i_s.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first interface springback in the model.

Arguments

- **Model** (Model)

Model to get first interface springback in

Returns

InterfaceSpringback object (or null if there are no interface springbacks in the model).

Return type

InterfaceSpringback

Example

To get the first interface springback in model m:

```
var i_s = InterfaceSpringback.First(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the interface springbacks in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all interface springbacks will be flagged in

- **flag** (Flag)

Flag to set on the interface springbacks

Returns

No return value

Example

To flag all of the interface springbacks with flag f in model m:

```
InterfaceSpringback.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the interface springback is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the interface springback

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if interface springback i_s has flag f set on it:

```
if (i_s.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each interface springback in the model.
Note that ForEach has been designed to make looping over interface springbacks as fast as possible and so has some limitations.
Firstly, a single temporary InterfaceSpringback object is created and on each function call it is updated with the current interface springback data. This means that you should not try to store the InterfaceSpringback object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new interface springbacks inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all interface springbacks are in

- **func** (function)

Function to call for each interface springback

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the interface springbacks in model m:

```
InterfaceSpringback.ForEach(m, test);
function test(i_s)
{
// i_s is InterfaceSpringback object
}
```

To call function test for all of the interface springbacks in model m with optional object:

```
var data = { x:0, y:0 };
InterfaceSpringback.ForEach(m, test, data);
function test(i_s, extra)
{
// i_s is InterfaceSpringback object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of InterfaceSpringback objects or properties for all of the interface springbacks in a model in PRIMER. If the optional property argument is not given then an array of InterfaceSpringback objects is returned. If the property argument is given, that property value for each interface springback is returned in the array instead of a InterfaceSpringback object

Arguments

- **Model** (Model)

Model to get interface springbacks from

- **property (optional)** (string)

Name for property to get for all interface springbacks in the model

Returns

Array of InterfaceSpringback objects or properties

Return type

Array

Example

To make an array of InterfaceSpringback objects for all of the interface springbacks in model m:

```
var a = InterfaceSpringback.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each interface springback in model m:

```
var a = InterfaceSpringback.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a interface springback.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the interface springback i_s:

```
var comm_array = i_s.GetComments();
```

GetExcludeKeyword(idx*[integer]*)

Description

Returns the keyword string excluded at given index in Keyword list. Needed only for InterfaceSpringback.EXCLUDE.

Arguments

- **idx** (integer)

The index in Keyword list you want the Keyword string for. **Note that indices start at 0, not 1**.

Returns

A Keyword string at index "idx" from excluded keyword list .

Return type

String

Example

To get the 3rd Keyword string in Interface Springback i_s:

```
if (i_s.nexclude >= 3)
{
    var keyword = i_s.GetExcludeKeyword(2);
}
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of InterfaceSpringback objects for all of the flagged interface springbacks in a model in PRIMER If the optional property argument is not given then an array of InterfaceSpringback objects is returned. If the property argument is given, then that property value for each interface springback is returned in the array instead of a InterfaceSpringback object

Arguments

- **Model** (Model)

Model to get interface springbacks from

- **flag** (Flag)

Flag set on the interface springbacks that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged interface springbacks in the model

Returns

Array of InterfaceSpringback objects or properties

Return type

Array

Example

To make an array of InterfaceSpringback objects for all of the interface springbacks in model m flagged with f:

```
var i_s = InterfaceSpringback.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the interface springbacks in model m flagged with f:

```
var a = InterfaceSpringback.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the InterfaceSpringback object for a interface springback ID.

Arguments

- **Model** (Model)

Model to find the interface springback in

- **number** (integer)

number of the interface springback you want the InterfaceSpringback object for

Returns

InterfaceSpringback object (or null if interface springback does not exist).

Return type

InterfaceSpringback

Example

To get the InterfaceSpringback object for interface springback 100 in model m

```
var i_s = InterfaceSpringback.GetFromID(m, 100);
```

GetNodalPoint(npt*[integer]*)

Description

Returns the data for nodal point constrained for *INTERFACE_SPRINGBACK.

Arguments

- **npt** (integer)

The nodal point you want the data for. **Note that nodal points start at 0, not 1**.

Returns

An array containing the Node id, translational constraint (TC) and rotational constraint (RC) constants.

Return type

Array

Example

To get the nodal point data for the 3rd nodal constraint for Interface Springback i_s:

```
if (i_s.nnodes >= 3)
{
    var npt_data = i_s.GetNodalPoint(2);
}
```

GetParameter(prop*[string]*)

Description

Checks if a InterfaceSpringback property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the InterfaceSpringback.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

interface springback property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if InterfaceSpringback property i_s.example is a parameter:

```
Options.property_parameter_names = true;
if (i_s.GetParameter(i_s.example) ) do_something...
Options.property_parameter_names = false;
```

To check if InterfaceSpringback property i_s.example is a parameter by using the GetParameter method:

```
if (i_s.ViewParameters().GetParameter(i_s.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this Interface Springback (*INTERFACE_SPRINGBACK_xxxx_xxxx) **Note that a carriage return is not added**. See also InterfaceSpringback.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for InterfaceSpringback i_s:

```
var key = i_s.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the InterfaceSpringback. **Note that a carriage return is not added**. See also InterfaceSpringback.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for InterfaceSpringback i_s:

```
var cards = i_s.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last interface springback in the model.

Arguments

- **Model** (Model)

Model to get last interface springback in

Returns

InterfaceSpringback object (or null if there are no interface springbacks in the model).

Return type

InterfaceSpringback

Example

To get the last interface springback in model m:

```
var i_s = InterfaceSpringback.Last(m);
```

Next()

Description

Returns the next interface springback in the model.

Arguments

No arguments

Returns

InterfaceSpringback object (or null if there are no more interface springbacks in the model).

Return type

InterfaceSpringback

Example

To get the interface springback in model m after interface springback i_s:

```
var i_s = i_s.Next();
```

Previous()

Description

Returns the previous interface springback in the model.

Arguments

No arguments

Returns

InterfaceSpringback object (or null if there are no more interface springbacks in the model).

Return type

InterfaceSpringback

Example

To get the interface springback in model m before interface springback i_s:

```
var i_s = i_s.Previous();
```

RemoveExcludeKeyword(idx*[integer]*)

Description

Removes the keyword string excluded at given index in Keyword list. Needed only for InterfaceSpringback.EXCLUDE

Arguments

- **idx** (integer)

The index in Keyword list you removed. **Note that indices start at 0, not 1**.

Returns

No return value.

Example

To remove the 3rd Keyword string in Interface Springback i_s:

```
if (i_s.nexclude >= 3)
{
    var keyword = i_s.RemoveExcludeKeyword(2);
}
```

RemoveNodalPoint(npt*[integer]*)

Description

Removes the nodal point for constrained node for *INTERFACE_SPRINGBACK.

Arguments

- **npt** (integer)

The nodal point you want to remove. **Note that nodal points start at 0, not 1**.

Returns

No return value.

Example

To remove the nodal point for the 3rd node for InterfaceSpringback i_s:

```
i_s.RemoveNodalPoint(2);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select interface springbacks using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting interface springbacks

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only interface springbacks from that model can be selected. If the argument is a Flag then only interface springbacks that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any interface springbacks can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of interface springbacks selected or null if menu cancelled

Return type

Number

Example

To select interface springbacks from model m, flagging those selected with flag f, giving the prompt 'Select interface springbacks':

```
InterfaceSpringback.Select(f, 'Select interface springbacks', m);
```

To select interface springbacks, flagging those selected with flag f but limiting selection to interface springbacks flagged with flag l, giving the prompt 'Select interface springbacks':

```
InterfaceSpringback.Select(f, 'Select interface springbacks', l);
```

SetExcludeKeyword(keystr*[string]*, index (optional)*[integer]*)

Description

Sets a keyword string to be excluded. Adds a new keyword if index value is not given, else replaces the keyword string at given index. **Note that indices start at 0, not 1**. Needed only for InterfaceSpringback.EXCLUDE

Arguments

- **keystr** (string)

The keyword string you want to be excluded.

- **index (optional)** (integer)

The index at which keyword string should be set.

Returns

No return value.

Example

To set a keyword string at index 3 to be excluded for InterfaceSpringback i_s:

```
if(i_s.nexclude >= 4)
    i_s.SetExcludeKeyword("ELEMENT_SHELL", 3);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the interface springback.

Arguments

- **flag** (Flag)

Flag to set on the interface springback

Returns

No return value

Example

To set flag f for interface springback i_s:

```
i_s.SetFlag(f);
```

SetNodalPoint(npt*[integer]*, nid*[integer]*, tc*[real]*, rc*[real]*)

Description

Sets the nodal point data for a node in *INTERFACE_SPRINGBACK.

Arguments

- **npt** (integer)

The nodal point you want to set the data for. **Note that nodal points start at 0, not 1**.

- **nid** (integer)

Node ID for the nodal point.

- **tc** (real)

Translational constraint constant of the nodal point. (0-7)

- **rc** (real)

Rotational constraint constant of the nodal point. (0-7)

Returns

No return value.

Example

To set the nodal data for the 3rd nodal point to node 1, tc 2 and rc 4, for InterfaceSpringback i_s:

```
i_s.SetNodalPoint(2, 1, 2, 4);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of interface springbacks in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing interface springbacks should be counted. If false or omitted referenced but undefined interface springbacks will also be included in the total.

Returns

number of interface springbacks

Return type

Number

Example

To get the total number of interface springbacks in model m:

```
var total = InterfaceSpringback.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the interface springbacks in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all interface springbacks will be unset in

- **flag** (Flag)

Flag to unset on the interface springbacks

Returns

No return value

Example

To unset the flag f on all the interface springbacks in model m:

```
InterfaceSpringback.UnflagAll(m, f);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

InterfaceSpringback object.

Return type

InterfaceSpringback

Example

To check if InterfaceSpringback property i_s.example is a parameter by using the InterfaceSpringback.GetParameter() method:

```
if (i_s.ViewParameters().GetParameter(i_s.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for interface springback with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The interface springback object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for interface springback i_s:

```
i_s.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for interface springback i_s:

```
i_s.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for interface springback. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for interface springback i_s:

```
i_s.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this interface springback.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for interface springback i_s:

```
var xrefs = i_s.Xrefs();
```

---

toString()

Description

Creates a string containing the InterfaceSpringback data in keyword format. Note that this contains the keyword header and the keyword cards. See also InterfaceSpringback.Keyword() and InterfaceSpringback.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for InterfaceSpringback i_s in keyword format

```
var i_str = i_s.toString();
```

LoadBeam class

The LoadBeam class gives you access to define load beam cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

LoadBeam constants

Name	Description
LoadBeam.ELEMENT	Load is *LOAD_BEAM_ELEMENT.
LoadBeam.SET	LOAD is *LOAD_BEAM_SET.

LoadBeam properties

Name	Type	Description
dal	integer	Direction of applied load. 1 for r-axis, 2 for s-axis or 3 for t-axis of beam.
eid	integer	NodeBeam ID or beam set ID. The esid property is an alternative name for this.
esid	integer	NodeBeam ID or beam set ID. The eid property is an alternative name for this.
exists (read only)	logical	true if load beam exists, false if referred to but not defined.
include	integer	The Include file number that the load beam is in.
lcid	integer	Curve ID or function ID
model (read only)	integer	The Model number that the load beam is in.
sf	real	Load curve scale factor
type	constant	The Load Beam type. Can be LoadBeam.ELEMENT or LoadBeam.SET.

Detailed Description

The LoadBeam class allows you to create, modify, edit and manipulate load beam cards. See the documentation below for more details.

Constructor

new LoadBeam(Model*[Model]*, type*[constant]*, eid/esid*[integer]*, dal*[integer]*, lcid*[integer]*, sf (optional)*[real]*)

Description

Create a new LoadBeam object.

Arguments

- **Model** (Model)

Model that load beam will be created in

- **type** (constant)

Specify the type of load beam (Can be LoadBeam.ELEMENT or LoadBeam.SET)

- **eid/esid** (integer)

Beam ID or beam set ID

- **dal** (integer)

Direction of applied load. 1 for r-axis, 2 for s-axis or 3 for t-axis of beam.

- **lcid** (integer)

Curve ID

- **sf (optional)** (real)

Load curve scale factor

Returns

LoadBeam object

Return type

LoadBeam

Example

To create a new load beam in model m, of type SET, with beam set 100, load parallel to s-axis, loadcurve 9 and a scale factor of 0.5:

```
var lb = new LoadBeam(m, LoadBeam.SET, 100, 2, 9, 0.5);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a load beam.

Arguments

- **Comment** (Comment)

Comment that will be attached to the load beam

Returns

No return value

Example

To associate comment c to the load beam lb:

```
lb.AssociateComment(c);
```

Blank()

Description

Blanks the load beam

Arguments

No arguments

Returns

No return value

Example

To blank load beam lb:

```
lb.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the load beams in the model.

Arguments

- **Model** (Model)

Model that all load beams will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the load beams in model m:

```
LoadBeam.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged load beams in the model.

Arguments

- **Model** (Model)

Model that all the flagged load beams will be blanked in

- **flag** (Flag)

Flag set on the load beams that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the load beams in model m flagged with f:

```
LoadBeam.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the load beam is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if load beam lb is blanked:

```
if (lb.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse load beam lb:

```
lb.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the load beam.

Arguments

- **flag** (Flag)

Flag to clear on the load beam

Returns

No return value

Example

To clear flag f for load beam lb:

```
lb.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the load beam. The target include of the copied load beam can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

LoadBeam object

Return type

LoadBeam

Example

To copy load beam lb into load beam z:

```
var z = lb.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a load beam

Arguments

- **Model** (Model)

Model that the load beam will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

LoadBeam object (or null if not made).

Return type

LoadBeam

Example

To start creating a load beam in model m:

```
var lb = LoadBeam.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a load beam.

Arguments

- **Comment** (Comment)

Comment that will be detached from the load beam

Returns

No return value

Example

To detach comment c from the load beam lb:

```
lb.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit load beam lb:

```
lb.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for load beam with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The load beam object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for load beam lb:

```
lb.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for load beam lb:

```
lb.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for load beam. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for load beam lb:

```
lb.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first load beam in the model.

Arguments

- **Model** (Model)

Model to get first load beam in

Returns

LoadBeam object (or null if there are no load beams in the model).

Return type

LoadBeam

Example

To get the first load beam in model m:

```
var lb = LoadBeam.First(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the load beams in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all load beams will be flagged in

- **flag** (Flag)

Flag to set on the load beams

Returns

No return value

Example

To flag all of the load beams with flag f in model m:

```
LoadBeam.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the load beam is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the load beam

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if load beam lb has flag f set on it:

```
if (lb.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each load beam in the model.
Note that ForEach has been designed to make looping over load beams as fast as possible and so has some limitations.
Firstly, a single temporary LoadBeam object is created and on each function call it is updated with the current load beam data. This means that you should not try to store the LoadBeam object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new load beams inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all load beams are in

- **func** (function)

Function to call for each load beam

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the load beams in model m:

```
LoadBeam.ForEach(m, test);
function test(lb)
{
// lb is LoadBeam object
}
```

To call function test for all of the load beams in model m with optional object:

```
var data = { x:0, y:0 };
LoadBeam.ForEach(m, test, data);
function test(lb, extra)
{
// lb is LoadBeam object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of LoadBeam objects or properties for all of the load beams in a model in PRIMER. If the optional property argument is not given then an array of LoadBeam objects is returned. If the property argument is given, that property value for each load beam is returned in the array instead of a LoadBeam object

Arguments

- **Model** (Model)

Model to get load beams from

- **property (optional)** (string)

Name for property to get for all load beams in the model

Returns

Array of LoadBeam objects or properties

Return type

Array

Example

To make an array of LoadBeam objects for all of the load beams in model m:

```
var a = LoadBeam.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each load beam in model m:

```
var a = LoadBeam.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a load beam.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the load beam lb:

```
var comm_array = lb.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of LoadBeam objects for all of the flagged load beams in a model in PRIMER If the optional property argument is not given then an array of LoadBeam objects is returned. If the property argument is given, then that property value for each load beam is returned in the array instead of a LoadBeam object

Arguments

- **Model** (Model)

Model to get load beams from

- **flag** (Flag)

Flag set on the load beams that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged load beams in the model

Returns

Array of LoadBeam objects or properties

Return type

Array

Example

To make an array of LoadBeam objects for all of the load beams in model m flagged with f:

```
var lb = LoadBeam.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the load beams in model m flagged with f:

```
var a = LoadBeam.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the LoadBeam object for a load beam ID.

Arguments

- **Model** (Model)

Model to find the load beam in

- **number** (integer)

number of the load beam you want the LoadBeam object for

Returns

LoadBeam object (or null if load beam does not exist).

Return type

LoadBeam

Example

To get the LoadBeam object for load beam 100 in model m

```
var lb = LoadBeam.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a LoadBeam property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the LoadBeam.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

load beam property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if LoadBeam property lb.example is a parameter:

```
Options.property_parameter_names = true;
if (lb.GetParameter(lb.example) ) do_something...
Options.property_parameter_names = false;
```

To check if LoadBeam property lb.example is a parameter by using the GetParameter method:

```
if (lb.ViewParameters().GetParameter(lb.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this load beam (*LOAD_BEAM_xxxx). **Note that a carriage return is not added**. See also LoadBeam.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for load beam lb:

```
var key = lb.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the load beam. **Note that a carriage return is not added**. See also LoadBeam.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for load beam lb:

```
var cards = lb.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last load beam in the model.

Arguments

- **Model** (Model)

Model to get last load beam in

Returns

LoadBeam object (or null if there are no load beams in the model).

Return type

LoadBeam

Example

To get the last load beam in model m:

```
var lb = LoadBeam.Last(m);
```

Next()

Description

Returns the next load beam in the model.

Arguments

No arguments

Returns

LoadBeam object (or null if there are no more load beams in the model).

Return type

LoadBeam

Example

To get the load beam in model m after load beam lb:

```
var lb = lb.Next();
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a load beam.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only load beams from that model can be picked. If the argument is a Flag then only load beams that are flagged with *limit* can be selected. If omitted, or null, any load beams from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

LoadBeam object (or null if not picked)

Return type

LoadBeam

Example

To pick a load beam from model m giving the prompt 'Pick load beam from screen':

```
var lb = LoadBeam.Pick('Pick load beam from screen', m);
```

Previous()

Description

Returns the previous load beam in the model.

Arguments

No arguments

Returns

LoadBeam object (or null if there are no more load beams in the model).

Return type

LoadBeam

Example

To get the load beam in model m before load beam lb:

```
var lb = lb.Previous();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select load beams using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting load beams

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only load beams from that model can be selected. If the argument is a Flag then only load beams that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any load beams can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of load beams selected or null if menu cancelled

Return type

Number

Example

To select load beams from model m, flagging those selected with flag f, giving the prompt 'Select load beams':

```
LoadBeam.Select(f, 'Select load beams', m);
```

To select load beams, flagging those selected with flag f but limiting selection to load beams flagged with flag l, giving the prompt 'Select load beams':

```
LoadBeam.Select(f, 'Select load beams', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the load beam.

Arguments

- **flag** (Flag)

Flag to set on the load beam

Returns

No return value

Example

To set flag f for load beam lb:

```
lb.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the load beam. The load beam will be sketched until you either call LoadBeam.Unsketch(), LoadBeam.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the load beam is sketched. If omitted redraw is true. If you want to sketch several load beams and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch load beam lb:

```
lb.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged load beams in the model. The load beams will be sketched until you either call LoadBeam.Unsketch(), LoadBeam.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged load beams will be sketched in

- **flag** (Flag)

Flag set on the load beams that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the load beams are sketched. If omitted redraw is true. If you want to sketch flagged load beams several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all load beams flagged with flag in model m:

```
LoadBeam.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of load beams in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing load beams should be counted. If false or omitted referenced but undefined load beams will also be included in the total.

Returns

number of load beams

Return type

Number

Example

To get the total number of load beams in model m:

```
var total = LoadBeam.Total(m);
```

Unblank()

Description

Unblanks the load beam

Arguments

No arguments

Returns

No return value

Example

To unblank load beam lb:

```
lb.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the load beams in the model.

Arguments

- **Model** (Model)

Model that all load beams will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the load beams in model m:

```
LoadBeam.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged load beams in the model.

Arguments

- **Model** (Model)

Model that the flagged load beams will be unblanked in

- **flag** (Flag)

Flag set on the load beams that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the load beams in model m flagged with f:

```
LoadBeam.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the load beams in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all load beams will be unset in

- **flag** (Flag)

Flag to unset on the load beams

Returns

No return value

Example

To unset the flag f on all the load beams in model m:

```
LoadBeam.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the load beam.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the load beam is unsketched. If omitted redraw is true. If you want to unsketch several load beams and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch load beam lb:

```
lb.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all load beams.

Arguments

- **Model** (Model)

Model that all load beams will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the load beams are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all load beams in model m:

```
LoadBeam.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged load beams in the model.

Arguments

- **Model** (Model)

Model that all load beams will be unsketched in

- **flag** (Flag)

Flag set on the load beams that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the load beams are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all load beams flagged with flag in model m:

```
LoadBeam.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

LoadBeam object.

Return type

LoadBeam

Example

To check if LoadBeam property lb.example is a parameter by using the LoadBeam.GetParameter() method:

```
if (lb.ViewParameters().GetParameter(lb.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for load beam with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The load beam object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for load beam lb:

```
lb.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for load beam lb:

```
lb.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for load beam. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for load beam lb:

```
lb.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this load beam.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for load beam lb:

```
var xrefs = lb.Xrefs();
```

---

toString()

Description

Creates a string containing the load beam data in keyword format. Note that this contains the keyword header and the keyword cards. See also LoadBeam.Keyword() and LoadBeam.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for load beam lb in keyword format

```
var s = lb.toString();
```

LoadBody class

The LoadBody class gives you access to *LOAD_BODY cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

LoadBody properties

Name	Type	Description
parts	Object	*LOAD_BODY_PARTS card
rx	Object	*LOAD_BODY_RX card
ry	Object	*LOAD_BODY_RY card
rz	Object	*LOAD_BODY_RZ card
vector	Object	*LOAD_BODY_VECTOR card
x	Object	*LOAD_BODY_X card
y	Object	*LOAD_BODY_Y card
z	Object	*LOAD_BODY_Z card

Properties for *LOAD_BODY_PARTS

Name	Type	Description
exists	logical	true if LoadBody card exists
include	integer	The Include file number that the LoadBody card is in
psid	integer	Part set id

Properties for *LOAD_BODY_RX

Name	Type	Description
cid	integer	Coordinate system
exists	logical	true if LoadBody card exists
include	integer	The Include file number that the LoadBody card is in
lcid	integer	Load curve ID
lciddr	integer	Load curve ID for dynamic relaxation
sf	real	Load curve scale factor
xc	real	X centre of rotation
yc	real	Y centre of rotation
zc	real	Z centre of rotation

Properties for *LOAD_BODY_RY

Name	Type	Description
cid	integer	Coordinate system
exists	logical	true if LoadBody card exists
include	integer	The Include file number that the LoadBody card is in
lcid	integer	Load curve ID
lciddr	integer	Load curve ID for dynamic relaxation
sf	real	Load curve scale factor
xc	real	X centre of rotation
yc	real	Y centre of rotation
zc	real	Z centre of rotation

Properties for *LOAD_BODY_RZ

Name	Type	Description
cid	integer	Coordinate system
exists	logical	true if LoadBody card exists
include	integer	The Include file number that the LoadBody card is in
lcid	integer	Load curve ID
lciddr	integer	Load curve ID for dynamic relaxation
sf	real	Load curve scale factor
xc	real	X centre of rotation
yc	real	Y centre of rotation
zc	real	Z centre of rotation

Properties for *LOAD_BODY_VECTOR

Name	Type	Description
cid	integer	Coordinate system
exists	logical	true if LoadBody card exists
include	integer	The Include file number that the LoadBody card is in
lcid	integer	Load curve ID
lciddr	integer	Load curve ID for dynamic relaxation
sf	real	Load curve scale factor
v1	real	X-component of Vector
v2	real	Y-component of Vector
v3	real	Z-component of Vector
xc	real	X centre of rotation
yc	real	Y centre of rotation
zc	real	Z centre of rotation

Properties for *LOAD_BODY_X

Name	Type	Description

cid	integer	Coordinate system
exists	logical	true if LoadBody card exists
include	integer	The Include file number that the LoadBody card is in
lcid	integer	Load curve ID
lciddr	integer	Load curve ID for dynamic relaxation
sf	real	Load curve scale factor
xc	real	X centre of rotation
yc	real	Y centre of rotation
zc	real	Z centre of rotation

Properties for *LOAD_BODY_Y

Name	Type	Description
cid	integer	Coordinate system
exists	logical	true if LoadBody card exists
include	integer	The Include file number that the LoadBody card is in
lcid	integer	Load curve ID
lciddr	integer	Load curve ID for dynamic relaxation
sf	real	Load curve scale factor
xc	real	X centre of rotation
yc	real	Y centre of rotation
zc	real	Z centre of rotation

Properties for *LOAD_BODY_Z

Name	Type	Description
cid	integer	Coordinate system
exists	logical	true if LoadBody card exists
include	integer	The Include file number that the LoadBody card is in
lcid	integer	Load curve ID
lciddr	integer	Load curve ID for dynamic relaxation
sf	real	Load curve scale factor
xc	real	X centre of rotation
yc	real	Y centre of rotation
zc	real	Z centre of rotation

Detailed Description

The LoadBody class allows you to create, modify, edit and manipulate *LOAD_BODY cards. Unlike other classes there is no constructor and there are no functions. Instead a LoadBody object is available as the loadBody property of a Model object. This object allows you to access all of the *LOAD_BODY cards.
For example, to activate *LOAD_BODY_X in model m and set lcid to 1.

```
m.loadBody.x.exists = true;
m.loadBody.x.lcid = 1;
```

See the properties for more details.

LoadBodyGeneralized class

The LoadBodyGeneralized class gives you access to define load body generalized cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model *[Model]*, redraw (optional) *[boolean]*)
- BlankFlagged(Model *[Model]*, flag *[Flag]*, redraw (optional) *[boolean]*)
- First(Model *[Model]*)
- FlagAll(Model *[Model]*, flag *[Flag]*)
- ForEach(Model *[Model]*, func *[function]*, extra (optional) *[any]*)
- GetAll(Model *[Model]*, property (optional) *[string]*)
- GetFlagged(Model *[Model]*, flag *[Flag]*, property (optional) *[string]*)
- GetFromID(Model *[Model]*, number *[integer]*)
- Last(Model *[Model]*)
- Pick(prompt *[string]*, limit (optional) *[Model or Flag]*, modal (optional) *[boolean]*, button text (optional) *[string]*)
- Select(flag *[Flag]*, prompt *[string]*, limit (optional) *[Model or Flag]*, modal (optional) *[boolean]*)
- SketchFlagged(Model *[Model]*, flag *[Flag]*, redraw (optional) *[boolean]*)
- Total(Model *[Model]*, exists (optional) *[boolean]*)
- UnblankAll(Model *[Model]*, redraw (optional) *[boolean]*)
- UnblankFlagged(Model *[Model]*, flag *[Flag]*, redraw (optional) *[boolean]*)
- UnflagAll(Model *[Model]*, flag *[Flag]*)
- UnsketchAll(Model *[Model]*, redraw (optional) *[boolean]*)
- UnsketchFlagged(Model *[Model]*, flag *[Flag]*, redraw (optional) *[boolean]*)

Member functions

- AssociateComment(Comment *[Comment]*)
- Blank()
- Blanked()
- ClearFlag(flag *[Flag]*)
- Copy(range (optional) *[boolean]*)
- DetachComment(Comment *[Comment]*)
- Error(message *[string]*, options (optional) *[object]*)
- Error(message *[string]*, details (optional) *[string]*) [deprecated]
- Flagged(flag *[Flag]*)
- GetComments()
- GetParameter(prop *[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag *[Flag]*)
- Sketch(redraw (optional) *[boolean]*)
- Unblank()
- Unsketch(redraw (optional) *[boolean]*)
- ViewParameters()
- Warning(message *[string]*, options (optional) *[object]*)
- Warning(message *[string]*, details (optional) *[string]*) [deprecated]
- Xrefs()
- toString()

LoadBodyGeneralized constants

Name	Description
LoadBodyGeneralized.NODE	Load is *LOAD_BODY_GENERALIZED.
LoadBodyGeneralized.SET_NODE	Load is *LOAD_BODY_GENERALIZED_SET_NODE.
LoadBodyGeneralized.SET_PART	LOAD is *LOAD_BODY_GENERALIZED_SET_PART.

LoadBodyGeneralized properties

Name	Type	Description
angtyp	string	Type of body loads
ax	real	Scale factor for acceleration in x-direction
ay	real	Scale factor for acceleration in y-direction
az	real	Scale factor for acceleration in z-direction
cid	integer	Coordinate system ID to define acceleration
drlcid	real	Curve ID for dynamic relaxation phase
exists (read only)	logical	true if load body generalized exists, false if referred to but not defined.
include	integer	The Include file number that the load body generalized is in.
lcid	integer	Curve ID
model (read only)	integer	The Model number that the load body generalized is in.
n1	integer	Beginning Node ID for body force load or the node or Part set ID
n2	integer	Ending Node ID for body force load. Set to zero if a set ID is defined
omx	real	Scale factor for x-angular velocity or acceleration
omy	real	Scale factor for y-angular velocity or acceleration
omz	real	Scale factor for z-angular velocity or acceleration
type	constant	The Load Node type, can be LoadBodyGeneralized.NODE or LoadBodyGeneralized.SET_NODE or LoadBodyGeneralized.SET_PART.
xc	real	X-center of rotation
yc	real	Y-center of rotation
zc	real	Z-center of rotation

Detailed Description

The LoadBodyGeneralized class allows you to create, modify, edit and manipulate load body generalized cards. See the documentation below for more details.

Constructor

new LoadBodyGeneralized(Model*[Model]*, type*[constant]*, n1*[integer]*, n2*[integer]*, lcid*[integer]*, drlcid (optional)*[integer]*, xc (optional)*[real]*, yc (optional)*[real]*, zc (optional)*[real]*, ax (optional)*[real]*, ay (optional)*[real]*, az (optional)*[real]*, omx (optional)*[real]*, omy (optional)*[real]*, omz (optional)*[real]*, cid (optional)*[integer]*, angtyp (optional)*[string]*)

Description

Create a new LoadBodyGeneralized object.

Arguments

- **Model** (Model)

Model that load body generalized will be created in

- **type** (constant)

Specify the type of load body generalized (Can be LoadBodyGeneralized.NODE or LoadBodyGeneralized.SET_NODE or LoadBodyGeneralized.SET_PART)

- **n1** (integer)

Beginning Node ID for body force load or the node or Part set ID

- **n2** (integer)

Ending Node ID for body force load. Set to zero if a set ID is defined

- **lcid** (integer)

Curve ID

- **drlcid (optional)** (integer)

Curve ID for dynamic relaxation phase

- **xc (optional)** (real)

X-center of rotation

- **yc (optional)** (real)

Y-center of rotation

- **zc (optional)** (real)

Z-center of rotation

- **ax (optional)** (real)

Scale factor for acceleration in x-direction

- **ay (optional)** (real)

Scale factor for acceleration in y-direction

- **az (optional)** (real)

Scale factor for acceleration in z-direction

- **omx (optional)** (real)

Scale factor for x-angular velocity or acceleration

- **omy (optional)** (real)

Scale factor for y-angular velocity or acceleration

- **omz (optional)** (real)

Scale factor for z-angular velocity or acceleration

- **cid (optional)** (integer)

Coordinate system ID to define acceleration

- **angtyp (optional)** (string)

Type of body loads

Returns

LoadBodyGeneralized object

Return type

LoadBodyGeneralized

Example

To create a new load body generalized in model m, of type SET_NODE, with LCID 9 and N2 is 2

```
var b = new LoadBodyGeneralized(m, LoadBodyGeneralized.SET_NODE, 100, 2, 9);
```

Details of functions

AssociateComment(Comment[*Comment*])

Description

Associates a comment with a load body generalized.

Arguments

- **Comment** (Comment)

Comment that will be attached to the load body generalized

Returns

No return value

Example

To associate comment c to the load body generalized lbg:

```
lbg.AssociateComment(c);
```

Blank()

Description

Blanks the load body generalized

Arguments

No arguments

Returns

No return value

Example

To blank load body generalized lbg:

```
lbg.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the load body generalizeds in the model.

Arguments

- **Model** (Model)

Model that all load body generalizeds will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the load body generalizeds in model m:

```
LoadBodyGeneralized.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged load body generalizeds in the model.

Arguments

- **Model** (Model)

Model that all the flagged load body generalizeds will be blanked in

- **flag** (Flag)

Flag set on the load body generalizeds that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the load body generalizeds in model m flagged with f:

```
LoadBodyGeneralized.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the load body generalized is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if load body generalized lbg is blanked:

```
if (lbg.Blanked() ) do_something...
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the load body generalized.

Arguments

- **flag** (Flag)

Flag to clear on the load body generalized

Returns

No return value

Example

To clear flag f for load body generalized lbg:

```
lbg.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the load body generalized. The target include of the copied load body generalized can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

LoadBodyGeneralized object

Return type

LoadBodyGeneralized

Example

To copy load body generalized lbg into load body generalized z:

```
var z = lbg.Copy();
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a load body generalized.

Arguments

- **Comment** (Comment)

Comment that will be detached from the load body generalized

Returns

No return value

Example

To detach comment c from the load body generalized lbg:

```
lbg.DetachComment(c);
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for load body generalized with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The load body generalized object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for load body generalized lbg:

```
lbg.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for load body generalized lbg:

```
lbg.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for load body generalized. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for load body generalized lbg:

```
lbg.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first load body generalized in the model.

Arguments

- **Model** (Model)

Model to get first load body generalized in

Returns

LoadBodyGeneralized object (or null if there are no load body generalizeds in the model).

Return type

LoadBodyGeneralized

Example

To get the first load body generalized in model m:

```
var lbg = LoadBodyGeneralized.First(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the load body generalizeds in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all load body generalizeds will be flagged in

- **flag** (Flag)

Flag to set on the load body generalizeds

Returns

No return value

Example

To flag all of the load body generalizeds with flag f in model m:

```
LoadBodyGeneralized.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the load body generalized is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the load body generalized

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if load body generalized lbg has flag f set on it:

```
if (lbg.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each load body generalized in the model.
Note that ForEach has been designed to make looping over load body generalizeds as fast as possible and so has some limitations.
Firstly, a single temporary LoadBodyGeneralized object is created and on each function call it is updated with the current load body generalized data. This means that you should not try to store the LoadBodyGeneralized object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new load body generalizeds inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all load body generalizeds are in

- **func** (function)

Function to call for each load body generalized

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the load body generalizeds in model m:

```
LoadBodyGeneralized.ForEach(m, test);
function test(lbg)
{
// lbg is LoadBodyGeneralized object
}
```

To call function test for all of the load body generalizeds in model m with optional object:

```
var data = { x:0, y:0 };
LoadBodyGeneralized.ForEach(m, test, data);
function test(lbg, extra)
{
// lbg is LoadBodyGeneralized object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of LoadBodyGeneralized objects or properties for all of the load body generalizeds in a model in PRIMER. If the optional property argument is not given then an array of LoadBodyGeneralized objects is returned. If the property argument is given, that property value for each load body generalized is returned in the array instead of a LoadBodyGeneralized object

Arguments

- **Model** (Model)

Model to get load body generalizeds from

- **property (optional)** (string)

Name for property to get for all load body generalizeds in the model

Returns

Array of LoadBodyGeneralized objects or properties

Return type

Array

Example

To make an array of LoadBodyGeneralized objects for all of the load body generalizeds in model m:

```
var a = LoadBodyGeneralized.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each load body generalized in model m:

```
var a = LoadBodyGeneralized.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a load body generalized.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the load body generalized lbg:

```
var comm_array = lbg.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of LoadBodyGeneralized objects for all of the flagged load body generalizeds in a model in PRIMER If the optional property argument is not given then an array of LoadBodyGeneralized objects is returned. If the property argument is given, then that property value for each load body generalized is returned in the array instead of a LoadBodyGeneralized object

Arguments

- **Model** (Model)

Model to get load body generalizeds from

- **flag** (Flag)

Flag set on the load body generalizeds that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged load body generalizeds in the model

Returns

Array of LoadBodyGeneralized objects or properties

Return type

Array

Example

To make an array of LoadBodyGeneralized objects for all of the load body generalizeds in model m flagged with f:

```
var lbg = LoadBodyGeneralized.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the load body generalizeds in model m flagged with f:

```
var a = LoadBodyGeneralized.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the LoadBodyGeneralized object for a load body generalized ID.

Arguments

- **Model** (Model)

Model to find the load body generalized in

- **number** (integer)

number of the load body generalized you want the LoadBodyGeneralized object for

Returns

LoadBodyGeneralized object (or null if load body generalized does not exist).

Return type

LoadBodyGeneralized

Example

To get the LoadBodyGeneralized object for load body generalized 100 in model m

```
var lbg = LoadBodyGeneralized.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a LoadBodyGeneralized property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the LoadBodyGeneralized.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

load body generalized property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if LoadBodyGeneralized property lbg.example is a parameter:

```
Options.property_parameter_names = true;
if (lbg.GetParameter(lbg.example) ) do_something...
Options.property_parameter_names = false;
```

To check if LoadBodyGeneralized property lbg.example is a parameter by using the GetParameter method:

```
if (lbg.ViewParameters().GetParameter(lbg.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this load body generalized (*LOAD_NODE_xxxx). **Note that a carriage return is not added**. See also LoadBodyGeneralized.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for load body generalized m:

```
var key = m.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the load body generalized. **Note that a carriage return is not added**. See also LoadBodyGeneralized.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for load body generalized l:

```
var cards = l.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last load body generalized in the model.

Arguments

- **Model** (Model)

Model to get last load body generalized in

Returns

LoadBodyGeneralized object (or null if there are no load body generalizeds in the model).

Return type

LoadBodyGeneralized

Example

To get the last load body generalized in model m:

```
var lbg = LoadBodyGeneralized.Last(m);
```

Next()

Description

Returns the next load body generalized in the model.

Arguments

No arguments

Returns

LoadBodyGeneralized object (or null if there are no more load body generalizeds in the model).

Return type

LoadBodyGeneralized

Example

To get the load body generalized in model m after load body generalized lbg:

```
var lbg = lbg.Next();
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a load body generalized.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only load body generalizeds from that model can be picked. If the argument is a Flag then only load body generalizeds that are flagged with *limit* can be selected. If omitted, or null, any load body generalizeds from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

LoadBodyGeneralized object (or null if not picked)

Return type

LoadBodyGeneralized

Example

To pick a load body generalized from model m giving the prompt 'Pick load body generalized from screen':

```
var lbg = LoadBodyGeneralized.Pick('Pick load body generalized from screen', m);
```

Previous()

Description

Returns the previous load body generalized in the model.

Arguments

No arguments

Returns

LoadBodyGeneralized object (or null if there are no more load body generalizeds in the model).

Return type

LoadBodyGeneralized

Example

To get the load body generalized in model m before load body generalized lbg:

```
var lbg = lbg.Previous();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select load body generalizeds using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting load body generalizeds

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only load body generalizeds from that model can be selected. If the argument is a Flag then only load body generalizeds that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any load body generalizeds can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of load body generalizeds selected or null if menu cancelled

Return type

Number

Example

To select load body generalizeds from model m, flagging those selected with flag f, giving the prompt 'Select load body generalizeds':

```
LoadBodyGeneralized.Select(f, 'Select load body generalizeds', m);
```

To select load body generalizeds, flagging those selected with flag f but limiting selection to load body generalizeds flagged with flag l, giving the prompt 'Select load body generalizeds':

```
LoadBodyGeneralized.Select(f, 'Select load body generalizeds', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the load body generalized.

Arguments

- **flag** (Flag)

Flag to set on the load body generalized

Returns

No return value

Example

To set flag f for load body generalized lbg:

```
lbg.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the load body generalized. The load body generalized will be sketched until you either call LoadBodyGeneralized.Unsketch(), LoadBodyGeneralized.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the load body generalized is sketched. If omitted redraw is true. If you want to sketch several load body generalizeds and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch load body generalized lbg:

```
lbg.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged load body generalizeds in the model. The load body generalizeds will be sketched until you either call LoadBodyGeneralized.Unsketch(), LoadBodyGeneralized.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged load body generalizeds will be sketched in

- **flag** (Flag)

Flag set on the load body generalizeds that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the load body generalizeds are sketched. If omitted redraw is true. If you want to sketch flagged load body generalizeds several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all load body generalizeds flagged with flag in model m:

```
LoadBodyGeneralized.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of load body generalizeds in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing load body generalizeds should be counted. If false or omitted referenced but undefined load body generalizeds will also be included in the total.

Returns

number of load body generalizeds

Return type

Number

Example

To get the total number of load body generalizeds in model m:

```
var total = LoadBodyGeneralized.Total(m);
```

Unblank()

Description

Unblanks the load body generalized

Arguments

No arguments

Returns

No return value

Example

To unblank load body generalized lbg:

```
lbg.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the load body generalizeds in the model.

Arguments

- **Model** (Model)

Model that all load body generalizeds will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the load body generalizeds in model m:

```
LoadBodyGeneralized.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged load body generalizeds in the model.

Arguments

- **Model** (Model)

Model that the flagged load body generalizeds will be unblanked in

- **flag** (Flag)

Flag set on the load body generalizeds that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the load body generalizeds in model m flagged with f:

```
LoadBodyGeneralized.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the load body generalizeds in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all load body generalizeds will be unset in

- **flag** (Flag)

Flag to unset on the load body generalizeds

Returns

No return value

Example

To unset the flag f on all the load body generalizeds in model m:

```
LoadBodyGeneralized.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the load body generalized.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the load body generalized is unsketched. If omitted redraw is true. If you want to unsketch several load body generalizeds and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch load body generalized lbg:

```
lbg.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all load body generalizeds.

Arguments

- **Model** (Model)

Model that all load body generalizeds will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the load body generalizeds are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all load body generalizeds in model m:

```
LoadBodyGeneralized.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged load body generalizeds in the model.

Arguments

- **Model** (Model)

Model that all load body generalizeds will be unsketched in

- **flag** (Flag)

Flag set on the load body generalizeds that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the load body generalizeds are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all load body generalizeds flagged with flag in model m:

```
LoadBodyGeneralized.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

LoadBodyGeneralized object.

Return type

LoadBodyGeneralized

Example

To check if LoadBodyGeneralized property lbg.example is a parameter by using the LoadBodyGeneralized.GetParameter() method:

```
if (lbg.ViewParameters().GetParameter(lbg.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for load body generalized with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The load body generalized object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for load body generalized lbg:

```
lbg.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for load body generalized lbg:

```
lbg.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for load body generalized. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for load body generalized lbg:

```
lbg.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this load body generalized.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for load body generalized lbg:

```
var xrefs = lbg.Xrefs();
```

---

toString()

Description

Creates a string containing the load body generalized data in keyword format. Note that this contains the keyword header and the keyword cards. See also LoadBodyGeneralized.Keyword() and LoadBodyGeneralized.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for load body generalized l in keyword format

```
var s = l.toString();
```

LoadGravity class

The LoadGravity class gives you access to define *LOAD_GRAVITY_PART cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- First(Model*[Model]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

LoadGravity constants

Name	Description

LoadGravity.PART	LOAD is *LOAD_GRAVITY_PART.
LoadGravity.SET_PART	LOAD is *LOAD_GRAVITY_PART_SET.

LoadGravity properties

Name	Type	Description
accel	real	Acceleration (will be multiplied by factor from curve)
dof	integer	Direction: enter 1, 2 or 3 for x, y, or z
exists (read only)	logical	true if load gravity exists, false if referred to but not defined.
include	integer	The Include file number that the load gravity is in.
lc	integer	Curve ID. Load curve defining factor vs. time (or zero if STGA, STGR are defined)
lcdr	integer	Curve ID. Load curve defining factor vs. time during dynamic relaxation
model (read only)	integer	The Model number that the load gravity is in.
pid	integer	Part ID or Part set ID
stga	integer	Construction Stages ID at which part is added (optional)
stgr	integer	Construction Stages ID at which part is removed (optional)
type	constant	The Load Gravity type. Can be LoadGravity.PART or LoadGravity.SET_PART.

Detailed Description

The LoadGravity class allows you to create, modify, edit and manipulate *LOAD_GRAVITY_PART cards. See the documentation below for more details.

Constructor

new LoadGravity(Model*[Model]*, type*[constant]*, pid*[integer]*, dof*[integer]*, lc*[integer]*, accel*[real]*, lcdr*[integer]*, stga (optional)*[integer]*, stgr (optional)*[integer]*)

Description

Create a new LoadGravity object.

Arguments

- **Model** (Model)

Model that LoadGravity will be created in

- **type** (constant)

Specify the type of LoadGravity (Can be LoadGravity.PART or LoadGravity.SET_PART

- **pid** (integer)

Part ID or Part set ID

- **dof** (integer)

Direction: enter 1, 2 or 3 for x, y or z

- **lc** (integer)

Curve ID. Load curve defining factor vs. time (or zero if STGA, STGR are defined)

- **accel** (real)

Acceleration (will be multiplied by factor from curve)

- **lcdr** (integer)

Curve ID. Load curve defining factor vs. time during dynamic relaxation

- **stga (optional)** (integer)

Construction Stage ID at which part is added

- **stgr (optional)** (integer)

Construction Stage ID at which part is removed

Returns

LoadGravity object

Return type

LoadGravity

Example

To create a new load gravity in model m, of type SET, with dof 2, loadcurve 9, acceleration of 0.5, and lcdr 10

```
var lg = new LoadGravity(m, LoadGravity.PART, 100, 2, 9, 0.5, 10);
```

Details of functions

AssociateComment(Comment[*Comment*])

Description

Associates a comment with a load gravity.

Arguments

- **Comment** (Comment)

Comment that will be attached to the load gravity

Returns

No return value

Example

To associate comment c to the load gravity lg:

```
lg.AssociateComment(c);
```

Blank()

Description

Blanks the load gravity

Arguments

No arguments

Returns

No return value

Example

To blank load gravity lg:

```
lg.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the load gravitys in the model.

Arguments

- **Model** (Model)

Model that all load gravitys will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the load gravitys in model m:

```
LoadGravity.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged load gravitys in the model.

Arguments

- **Model** (Model)

Model that all the flagged load gravitys will be blanked in

- **flag** (Flag)

Flag set on the load gravitys that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the load gravitys in model m flagged with f:

```
LoadGravity.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the load gravity is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if load gravity lg is blanked:

```
if (lg.Blanked() ) do_something...
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the load gravity.

Arguments

- **flag** (Flag)

Flag to clear on the load gravity

Returns

No return value

Example

To clear flag f for load gravity lg:

```
lg.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the load gravity. The target include of the copied load gravity can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

LoadGravity object

Return type

LoadGravity

Example

To copy load gravity lg into load gravity z:

```
var z = lg.Copy();
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a load gravity.

Arguments

- **Comment** (Comment)

Comment that will be detached from the load gravity

Returns

No return value

Example

To detach comment c from the load gravity lg:

```
lg.DetachComment(c);
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for load gravity with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The load gravity object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for load gravity lg:

```
lg.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for load gravity lg:

```
lg.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for load gravity. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for load gravity lg:

```
lg.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first load gravity in the model.

Arguments

- **Model** (Model)

Model to get first load gravity in

Returns

LoadGravity object (or null if there are no load gravitys in the model).

Return type

LoadGravity

Example

To get the first load gravity in model m:

```
var lg = LoadGravity.First(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the load gravitys in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all load gravitys will be flagged in

- **flag** (Flag)

Flag to set on the load gravitys

Returns

No return value

Example

To flag all of the load gravitys with flag f in model m:

```
LoadGravity.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the load gravity is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the load gravity

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if load gravity lg has flag f set on it:

```
if (lg.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each load gravity in the model.
Note that ForEach has been designed to make looping over load gravitys as fast as possible and so has some limitations.
Firstly, a single temporary LoadGravity object is created and on each function call it is updated with the current load gravity data. This means that you should not try to store the LoadGravity object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new load gravitys inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all load gravitys are in

- **func** (function)

Function to call for each load gravity

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the load gravitys in model m:

```
LoadGravity.ForEach(m, test);
function test(lg)
{
// lg is LoadGravity object
}
```

To call function test for all of the load gravitys in model m with optional object:

```
var data = { x:0, y:0 };
LoadGravity.ForEach(m, test, data);
function test(lg, extra)
{
// lg is LoadGravity object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of LoadGravity objects or properties for all of the load gravitys in a model in PRIMER. If the optional property argument is not given then an array of LoadGravity objects is returned. If the property argument is given, that property value for each load gravity is returned in the array instead of a LoadGravity object

Arguments

- **Model** (Model)

Model to get load gravitys from

- **property (optional)** (string)

Name for property to get for all load gravitys in the model

Returns

Array of LoadGravity objects or properties

Return type

Array

Example

To make an array of LoadGravity objects for all of the load gravitys in model m:

```
var a = LoadGravity.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each load gravity in model m:

```
var a = LoadGravity.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a load gravity.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the load gravity lg:

```
var comm_array = lg.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of LoadGravity objects for all of the flagged load gravitys in a model in PRIMER If the optional property argument is not given then an array of LoadGravity objects is returned. If the property argument is given, then that property value for each load gravity is returned in the array instead of a LoadGravity object

Arguments

- **Model** (Model)

Model to get load gravitys from

- **flag** (Flag)

Flag set on the load gravitys that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged load gravitys in the model

Returns

Array of LoadGravity objects or properties

Return type

Array

Example

To make an array of LoadGravity objects for all of the load gravitys in model m flagged with f:

```
var lg = LoadGravity.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the load gravitys in model m flagged with f:

```
var a = LoadGravity.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the LoadGravity object for a load gravity ID.

Arguments

- **Model** (Model)

Model to find the load gravity in

- **number** (integer)

number of the load gravity you want the LoadGravity object for

Returns

LoadGravity object (or null if load gravity does not exist).

Return type

LoadGravity

Example

To get the LoadGravity object for load gravity 100 in model m

```
var lg = LoadGravity.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a LoadGravity property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the LoadGravity.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

load gravity property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if LoadGravity property lg.example is a parameter:

```
Options.property_parameter_names = true;
if (lg.GetParameter(lg.example) ) do_something...
Options.property_parameter_names = false;
```

To check if LoadGravity property lg.example is a parameter by using the GetParameter method:

```
if (lg.ViewParameters().GetParameter(lg.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this LoadGravity (*LOAD_GRAVITY_PART). **Note that a carriage return is not added**. See also LoadGravity.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for LoadGravity lg:

```
var key = lg.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the LoadGravity. **Note that a carriage return is not added**. See also LoadGravity.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for LoadGravity lg:

```
var cards = lg.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last load gravity in the model.

Arguments

- **Model** (Model)

Model to get last load gravity in

Returns

LoadGravity object (or null if there are no load gravitys in the model).

Return type

LoadGravity

Example

To get the last load gravity in model m:

```
var lg = LoadGravity.Last(m);
```

Next()

Description

Returns the next load gravity in the model.

Arguments

No arguments

Returns

LoadGravity object (or null if there are no more load gravitys in the model).

Return type

LoadGravity

Example

To get the load gravity in model m after load gravity lg:

```
var lg = lg.Next();
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a load gravity.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only load gravitys from that model can be picked. If the argument is a Flag then only load gravitys that are flagged with *limit* can be selected. If omitted, or null, any load gravitys from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

LoadGravity object (or null if not picked)

Return type

LoadGravity

Example

To pick a load gravity from model m giving the prompt 'Pick load gravity from screen':

```
var lg = LoadGravity.Pick('Pick load gravity from screen', m);
```

Previous()

Description

Returns the previous load gravity in the model.

Arguments

No arguments

Returns

LoadGravity object (or null if there are no more load gravitys in the model).

Return type

LoadGravity

Example

To get the load gravity in model m before load gravity lg:

```
var lg = lg.Previous();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select load gravitys using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting load gravitys

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only load gravitys from that model can be selected. If the argument is a Flag then only load gravitys that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any load gravitys can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of load gravitys selected or null if menu cancelled

Return type

Number

Example

To select load gravitys from model m, flagging those selected with flag f, giving the prompt 'Select load gravitys':

```
LoadGravity.Select(f, 'Select load gravitys', m);
```

To select load gravitys, flagging those selected with flag f but limiting selection to load gravitys flagged with flag l, giving the prompt 'Select load gravitys':

```
LoadGravity.Select(f, 'Select load gravitys', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the load gravity.

Arguments

- **flag** (Flag)

Flag to set on the load gravity

Returns

No return value

Example

To set flag f for load gravity lg:

```
lg.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the load gravity. The load gravity will be sketched until you either call LoadGravity.Unsketch(), LoadGravity.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the load gravity is sketched. If omitted redraw is true. If you want to sketch several load gravitys and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch load gravity lg:

```
lg.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged load gravitys in the model. The load gravitys will be sketched until you either call LoadGravity.Unsketch(), LoadGravity.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged load gravitys will be sketched in

- **flag** (Flag)

Flag set on the load gravitys that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the load gravitys are sketched. If omitted redraw is true. If you want to sketch flagged load gravitys several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all load gravitys flagged with flag in model m:

```
LoadGravity.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of load gravitys in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing load gravitys should be counted. If false or omitted referenced but undefined load gravitys will also be included in the total.

Returns

number of load gravitys

Return type

Number

Example

To get the total number of load gravitys in model m:

```
var total = LoadGravity.Total(m);
```

Unblank()

Description

Unblanks the load gravity

Arguments

No arguments

Returns

No return value

Example

To unblank load gravity lg:

```
lg.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the load gravitys in the model.

Arguments

- **Model** (Model)

Model that all load gravitys will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the load gravitys in model m:

```
LoadGravity.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged load gravitys in the model.

Arguments

- **Model** (Model)

Model that the flagged load gravitys will be unblanked in

- **flag** (Flag)

Flag set on the load gravitys that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the load gravitys in model m flagged with f:

```
LoadGravity.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the load gravitys in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all load gravitys will be unset in

- **flag** (Flag)

Flag to unset on the load gravitys

Returns

No return value

Example

To unset the flag f on all the load gravitys in model m:

```
LoadGravity.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the load gravity.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the load gravity is unsketched. If omitted redraw is true. If you want to unsketch several load gravitys and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch load gravity lg:

```
lg.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all load gravitys.

Arguments

- **Model** (Model)

Model that all load gravitys will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the load gravitys are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all load gravitys in model m:

```
LoadGravity.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged load gravitys in the model.

Arguments

- **Model** (Model)

Model that all load gravitys will be unsketched in

- **flag** (Flag)

Flag set on the load gravitys that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the load gravitys are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all load gravitys flagged with flag in model m:

```
LoadGravity.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with ’method chaining’ (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

LoadGravity object.

Return type

LoadGravity

Example

To check if LoadGravity property lg.example is a parameter by using the LoadGravity.GetParameter() method:

```
if (lg.ViewParameters().GetParameter(lg.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for load gravity with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The load gravity object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for load gravity lg:

```
lg.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for load gravity lg:

```
lg.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for load gravity. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for load gravity lg:

```
lg.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this load gravity.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for load gravity lg:

```
var xrefs = lg.Xrefs();
```

---

toString()

Description

Creates a string containing the LoadGravity data in keyword format. Note that this contains the keyword header and the keyword cards. See also LoadGravity.Keyword() and LoadGravity.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for LoadGravity lg in keyword format

```
var s = lg.toString();
```

LoadNode class

The LoadNode class gives you access to define load node cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- First(Model*[Model]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

LoadNode constants

Name	Description

LoadNode.POINT	Load is *LOAD_NODE_POINT.
LoadNode.SET	LOAD is *LOAD_NODE_SET.
LoadNode.SET_ONCE	LOAD is *LOAD_NODE_SET_ONCE.

LoadNode properties

Name	Type	Description
cid	integer	Coordinate system ID
dof	integer	Applicable degrees-of-freedom
exists (read only)	logical	true if load node exists, false if referred to but not defined.
include	integer	The Include file number that the load node is in.
lcid	integer	Curve ID
m1	integer	Node 1 ID
m2	integer	Node 2 ID
m3	integer	Node 3 ID
model (read only)	integer	The Model number that the load node is in.
nid	integer	Node ID or node set ID
sf	real	Curve scale factor
type	constant	The Load Node type. Can be LoadNode.POINT or LoadNode.SET. LoadNode.SET_ONCE.

Detailed Description

The LoadNode class allows you to create, modify, edit and manipulate load node cards. See the documentation below for more details.

Constructor

new LoadNode(Model*[Model]*, type*[constant]*, nid*[integer]*, dof*[integer]*, lcid*[integer]*, sf (optional)*[real]*, cid (optional)*[integer]*, m1 (optional)*[integer]*, m2 (optional)*[integer]*, m3 (optional)*[integer]*, lcidsf (optional)*[integer]*)

Description

Create a new LoadNode object.

Arguments

- **Model** (Model)

Model that load node will be created in

- **type** (constant)

Specify the type of load node (Can be LoadNode.POINT or LoadNode.SET) LoadNode.SET_ONCE)

- **nid** (integer)

Node ID or node set ID

- **dof** (integer)

Applicable degrees-of-freedom

- **lcid** (integer)

Curve ID

- **sf (optional)** (real)

Curve scale factor

- **cid (optional)** (integer)

Coordinate system ID

- **m1 (optional)** (integer)

Node 1 ID

- **m2 (optional)** (integer)

Node 2 ID

- **m3 (optional)** (integer)

Node 3 ID

- **lcidsf (optional)** (integer)

Curve ID

Returns

LoadNode object

Return type

LoadNode

Example

To create a new load node in model m, of type SET, with loadcurve 9 and a scale factor of 0.5

```
var b = new LoadNode(m, LoadNode.SET, 100, 2, 9, 0.5);
```

Details of functions

AssociateComment(Comment[*Comment*])

Description

Associates a comment with a load node.

Arguments

- **Comment** (Comment)

Comment that will be attached to the load node

Returns

No return value

Example

To associate comment c to the load node ln:

```
ln.AssociateComment(c);
```

Blank()

Description

Blanks the load node

Arguments

No arguments

Returns

No return value

Example

To blank load node ln:

```
ln.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the load nodes in the model.

Arguments

- **Model** (Model)

Model that all load nodes will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the load nodes in model m:

```
LoadNode.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged load nodes in the model.

Arguments

- **Model** (Model)

Model that all the flagged load nodes will be blanked in

- **flag** (Flag)

Flag set on the load nodes that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the load nodes in model m flagged with f:

```
LoadNode.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the load node is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if load node ln is blanked:

```
if (ln.Blanked() ) do_something...
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the load node.

Arguments

- **flag** (Flag)

Flag to clear on the load node

Returns

No return value

Example

To clear flag f for load node ln:

```
ln.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the load node. The target include of the copied load node can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

LoadNode object

Return type

LoadNode

Example

To copy load node ln into load node z:

```
var z = ln.Copy();
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a load node.

Arguments

- **Comment** (Comment)

Comment that will be detached from the load node

Returns

No return value

Example

To detach comment c from the load node ln:

```
ln.DetachComment(c);
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for load node with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The load node object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for load node ln:

```
ln.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for load node ln:

```
ln.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for load node. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for load node ln:

```
ln.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first load node in the model.

Arguments

- **Model** (Model)

Model to get first load node in

Returns

LoadNode object (or null if there are no load nodes in the model).

Return type

LoadNode

Example

To get the first load node in model m:

```
var ln = LoadNode.First(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the load nodes in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all load nodes will be flagged in

- **flag** (Flag)

Flag to set on the load nodes

Returns

No return value

Example

To flag all of the load nodes with flag f in model m:

```
LoadNode.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the load node is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the load node

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if load node ln has flag f set on it:

```
if (ln.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each load node in the model.
Note that ForEach has been designed to make looping over load nodes as fast as possible and so has some limitations.
Firstly, a single temporary LoadNode object is created and on each function call it is updated with the current load node data. This means that you should not try to store the LoadNode object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new load nodes inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all load nodes are in

- **func** (function)

Function to call for each load node

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the load nodes in model m:

```
LoadNode.ForEach(m, test);
function test(ln)
{
// ln is LoadNode object
}
```

To call function test for all of the load nodes in model m with optional object:

```
var data = { x:0, y:0 };
LoadNode.ForEach(m, test, data);
function test(ln, extra)
{
// ln is LoadNode object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of LoadNode objects or properties for all of the load nodes in a model in PRIMER. If the optional property argument is not given then an array of LoadNode objects is returned. If the property argument is given, that property value for each load node is returned in the array instead of a LoadNode object

Arguments

- **Model** (Model)

Model to get load nodes from

- **property (optional)** (string)

Name for property to get for all load nodes in the model

Returns

Array of LoadNode objects or properties

Return type

Array

Example

To make an array of LoadNode objects for all of the load nodes in model m:

```
var a = LoadNode.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each load node in model m:

```
var a = LoadNode.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a load node.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the load node ln:

```
var comm_array = ln.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of LoadNode objects for all of the flagged load nodes in a model in PRIMER If the optional property argument is not given then an array of LoadNode objects is returned. If the property argument is given, then that property value for each load node is returned in the array instead of a LoadNode object

Arguments

- **Model** (Model)

Model to get load nodes from

- **flag** (Flag)

Flag set on the load nodes that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged load nodes in the model

Returns

Array of LoadNode objects or properties

Return type

Array

Example

To make an array of LoadNode objects for all of the load nodes in model m flagged with f:

```
var ln = LoadNode.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the load nodes in model m flagged with f:

```
var a = LoadNode.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the LoadNode object for a load node ID.

Arguments

- **Model** (Model)

Model to find the load node in

- **number** (integer)

number of the load node you want the LoadNode object for

Returns

LoadNode object (or null if load node does not exist).

Return type

LoadNode

Example

To get the LoadNode object for load node 100 in model m

```
var ln = LoadNode.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a LoadNode property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the LoadNode.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

load node property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if LoadNode property ln.example is a parameter:

```
Options.property_parameter_names = true;
if (ln.GetParameter(ln.example) ) do_something...
Options.property_parameter_names = false;
```

To check if LoadNode property ln.example is a parameter by using the GetParameter method:

```
if (ln.ViewParameters().GetParameter(ln.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this load node (*LOAD_NODE_xxxx). **Note that a carriage return is not added**. See also LoadNode.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for load node m:

```
var key = m.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the load node. **Note that a carriage return is not added**. See also LoadNode.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for load node l:

```
var cards = l.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last load node in the model.

Arguments

- **Model** (Model)

Model to get last load node in

Returns

LoadNode object (or null if there are no load nodes in the model).

Return type

LoadNode

Example

To get the last load node in model m:

```
var ln = LoadNode.Last(m);
```

Next()

Description

Returns the next load node in the model.

Arguments

No arguments

Returns

LoadNode object (or null if there are no more load nodes in the model).

Return type

LoadNode

Example

To get the load node in model m after load node ln:

```
var ln = ln.Next();
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a load node.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only load nodes from that model can be picked. If the argument is a Flag then only load nodes that are flagged with *limit* can be selected. If omitted, or null, any load nodes from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

LoadNode object (or null if not picked)

Return type

LoadNode

Example

To pick a load node from model m giving the prompt 'Pick load node from screen':

```
var ln = LoadNode.Pick('Pick load node from screen', m);
```

Previous()

Description

Returns the previous load node in the model.

Arguments

No arguments

Returns

LoadNode object (or null if there are no more load nodes in the model).

Return type

LoadNode

Example

To get the load node in model m before load node ln:

```
var ln = ln.Previous();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select load nodes using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting load nodes

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only load nodes from that model can be selected. If the argument is a Flag then only load nodes that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any load nodes can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of load nodes selected or null if menu cancelled

Return type

Number

Example

To select load nodes from model m, flagging those selected with flag f, giving the prompt 'Select load nodes':

```
LoadNode.Select(f, 'Select load nodes', m);
```

To select load nodes, flagging those selected with flag f but limiting selection to load nodes flagged with flag l, giving the prompt 'Select load nodes':

```
LoadNode.Select(f, 'Select load nodes', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the load node.

Arguments

- **flag** (Flag)

Flag to set on the load node

Returns

No return value

Example

To set flag f for load node ln:

```
ln.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the load node. The load node will be sketched until you either call LoadNode.Unsketch(), LoadNode.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the load node is sketched. If omitted redraw is true. If you want to sketch several load nodes and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch load node ln:

```
ln.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged load nodes in the model. The load nodes will be sketched until you either call LoadNode.Unsketch(), LoadNode.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged load nodes will be sketched in

- **flag** (Flag)

Flag set on the load nodes that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the load nodes are sketched. If omitted redraw is true. If you want to sketch flagged load nodes several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all load nodes flagged with flag in model m:

```
LoadNode.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of load nodes in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing load nodes should be counted. If false or omitted referenced but undefined load nodes will also be included in the total.

Returns

number of load nodes

Return type

Number

Example

To get the total number of load nodes in model m:

```
var total = LoadNode.Total(m);
```

Unblank()

Description

Unblanks the load node

Arguments

No arguments

Returns

No return value

Example

To unblank load node ln:

```
ln.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the load nodes in the model.

Arguments

- **Model** (Model)

Model that all load nodes will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the load nodes in model m:

```
LoadNode.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged load nodes in the model.

Arguments

- **Model** (Model)

Model that the flagged load nodes will be unblanked in

- **flag** (Flag)

Flag set on the load nodes that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the load nodes in model m flagged with f:

```
LoadNode.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the load nodes in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all load nodes will be unset in

- **flag** (Flag)

Flag to unset on the load nodes

Returns

No return value

Example

To unset the flag f on all the load nodes in model m:

```
LoadNode.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the load node.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the load node is unsketched. If omitted redraw is true. If you want to unsketch several load nodes and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch load node ln:

```
ln.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all load nodes.

Arguments

- **Model** (Model)

Model that all load nodes will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the load nodes are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all load nodes in model m:

```
LoadNode.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged load nodes in the model.

Arguments

- **Model** (Model)

Model that all load nodes will be unsketched in

- **flag** (Flag)

Flag set on the load nodes that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the load nodes are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all load nodes flagged with flag in model m:

```
LoadNode.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

LoadNode object.

Return type

LoadNode

Example

To check if LoadNode property ln.example is a parameter by using the LoadNode.GetParameter() method:

```
if (ln.ViewParameters().GetParameter(ln.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for load node with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The load node object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for load node ln:

```
ln.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for load node ln:

```
ln.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for load node. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for load node ln:

```
ln.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this load node.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for load node ln:

```
var xrefs = ln.Xrefs();
```

---

toString()

Description

Creates a string containing the load node data in keyword format. Note that this contains the keyword header and the keyword cards. See also LoadNode.Keyword() and LoadNode.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for load node l in keyword format

```
var s = l.toString();
```

LoadRemovePart class

The LoadRemovePart class gives you access to define *LOAD_REMOVE_PART cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- First(Model*[Model]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

LoadRemovePart constants

Name	Description

LoadRemovePart.PART	LOAD is *LOAD_REMOVE_PART.
LoadRemovePart.SET_PART	LOAD is *LOAD_REMOVE_PART_SET.

LoadRemovePart properties

Name	Type	Description
exists (read only)	logical	true if load remove_part exists, false if referred to but not defined.
include	integer	The Include file number that the load remove_part is in.
model (read only)	integer	The Model number that the load remove_part is in.
pid	integer	Part ID or Part Set ID
stgr	integer	Construction Stages ID at which part is removed.
time0	real	Time at which stress reduction starts.
time1	real	Time at which stresses become zero and elements are deleted.
type	constant	The Load RemovePart type. Can be LoadRemovePart.PART or LoadRemovePart.SET_PART.

Detailed Description

The LoadRemovePart class allows you to create, modify, edit and manipulate *LOAD_REMOVE_PART cards. See the documentation below for more details.

Constructor

new LoadRemovePart(Model*[Model]*, type*[constant]*, pid*[integer]*, time0 (optional)*[real]*, time1 (optional)*[real]*, stgr (optional)*[integer]*)

Description

Create a new LoadRemovePart object.

Arguments

- **Model** (Model)

Model that LoadRemovePart will be created in

- **type** (constant)

Specify the type of LoadRemovePart (Can be LoadRemovePart.PART or LoadRemovePart.SET_PART

- **pid** (integer)

Part ID or Part Set ID

- **time0 (optional)** (real)

Time at which stress reduction starts.

- **time1 (optional)** (real)

Time at which stresses become zero and elements are deleted.

- **stgr (optional)** (integer)

Construction Stage ID at which part is removed.

Returns

LoadRemovePart object

Return type

LoadRemovePart

Example

To create a new load remove_part in model m, of type PART, with pid 100, time0 2.5 and time1 4.5.

```
var l_r_p = new LoadRemovePart(m, LoadRemovePart.PART, 100, 2.5, 4.5);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a load remove_part.

Arguments

- **Comment** (Comment)

Comment that will be attached to the load remove_part

Returns

No return value

Example

To associate comment c to the load remove_part l_r_p:

```
l_r_p.AssociateComment(c);
```

Blank()

Description

Blanks the load remove_part

Arguments

No arguments

Returns

No return value

Example

To blank load remove_part l_r_p:

```
l_r_p.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the load remove_parts in the model.

Arguments

- **Model** (Model)

Model that all load remove_parts will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the load remove_parts in model m:

```
LoadRemovePart.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged load remove_parts in the model.

Arguments

- **Model** (Model)

Model that all the flagged load remove_parts will be blanked in

- **flag** (Flag)

Flag set on the load remove_parts that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the load remove_parts in model m flagged with f:

```
LoadRemovePart.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the load remove_part is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if load remove_part l_r_p is blanked:

```
if (l_r_p.Blanked() ) do_something...
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the load remove_part.

Arguments

- **flag** (Flag)

Flag to clear on the load remove_part

Returns

No return value

Example

To clear flag f for load remove_part l_r_p:

```
l_r_p.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the load remove_part. The target include of the copied load remove_part can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

LoadRemovePart object

Return type

LoadRemovePart

Example

To copy load remove_part l_r_p into load remove_part z:

```
var z = l_r_p.Copy();
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a load remove_part.

Arguments

- **Comment** (Comment)

Comment that will be detached from the load remove_part

Returns

No return value

Example

To detach comment c from the load remove_part l_r_p:

```
l_r_p.DetachComment(c);
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for load remove_part with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The load remove_part object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for load remove_part l_r_p:

```
l_r_p.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for load remove_part l_r_p:

```
l_r_p.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for load remove_part. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for load remove_part l_r_p:

```
l_r_p.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first load remove_part in the model.

Arguments

- **Model** (Model)

Model to get first load remove_part in

Returns

LoadRemovePart object (or null if there are no load remove_parts in the model).

Return type

LoadRemovePart

Example

To get the first load remove_part in model m:

```
var l_r_p = LoadRemovePart.First(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the load remove_parts in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all load remove_parts will be flagged in

- **flag** (Flag)

Flag to set on the load remove_parts

Returns

No return value

Example

To flag all of the load remove_parts with flag f in model m:

```
LoadRemovePart.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the load remove_part is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the load remove_part

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if load remove_part l_r_p has flag f set on it:

```
if (l_r_p.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each load remove_part in the model.
Note that ForEach has been designed to make looping over load remove_parts as fast as possible and so has some limitations.
Firstly, a single temporary LoadRemovePart object is created and on each function call it is updated with the current load remove_part data. This means that you should not try to store the LoadRemovePart object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new load remove_parts inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all load remove_parts are in

- **func** (function)

Function to call for each load remove_part

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the load remove_parts in model m:

```
LoadRemovePart.ForEach(m, test);
function test(l_r_p)
{
// l_r_p is LoadRemovePart object
}
```

To call function test for all of the load remove_parts in model m with optional object:

```
var data = { x:0, y:0 };
LoadRemovePart.ForEach(m, test, data);
function test(l_r_p, extra)
{
// l_r_p is LoadRemovePart object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of LoadRemovePart objects or properties for all of the load remove_parts in a model in PRIMER. If the optional property argument is not given then an array of LoadRemovePart objects is returned. If the property argument is given, that property value for each load remove_part is returned in the array instead of a LoadRemovePart object

Arguments

- **Model** (Model)

Model to get load remove_parts from

- **property (optional)** (string)

Name for property to get for all load remove_parts in the model

Returns

Array of LoadRemovePart objects or properties

Return type

Array

Example

To make an array of LoadRemovePart objects for all of the load remove_parts in model m:

```
var a = LoadRemovePart.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each load remove_part in model m:

```
var a = LoadRemovePart.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a load remove_part.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the load remove_part l_r_p:

```
var comm_array = l_r_p.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of LoadRemovePart objects for all of the flagged load remove_parts in a model in PRIMER If the optional property argument is not given then an array of LoadRemovePart objects is returned. If the property argument is given, then that property value for each load remove_part is returned in the array instead of a LoadRemovePart object

Arguments

- **Model** (Model)

Model to get load remove_parts from

- **flag** (Flag)

Flag set on the load remove_parts that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged load remove_parts in the model

Returns

Array of LoadRemovePart objects or properties

Return type

Array

Example

To make an array of LoadRemovePart objects for all of the load remove_parts in model m flagged with f:

```
var l_r_p = LoadRemovePart.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the load remove_parts in model m flagged with f:

```
var a = LoadRemovePart.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the LoadRemovePart object for a load remove_part ID.

Arguments

- **Model** (Model)

Model to find the load remove_part in

- **number** (integer)

number of the load remove_part you want the LoadRemovePart object for

Returns

LoadRemovePart object (or null if load remove_part does not exist).

Return type

LoadRemovePart

Example

To get the LoadRemovePart object for load remove_part 100 in model m

```
var l_r_p = LoadRemovePart.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a LoadRemovePart property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the LoadRemovePart.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

load remove_part property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if LoadRemovePart property l_r_p.example is a parameter:

```
Options.property_parameter_names = true;
if (l_r_p.GetParameter(l_r_p.example) ) do_something...
Options.property_parameter_names = false;
```

To check if LoadRemovePart property l_r_p.example is a parameter by using the GetParameter method:

```
if (l_r_p.ViewParameters().GetParameter(l_r_p.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this LoadRemovePart (*LOAD_REMOVE_PART). **Note that a carriage return is not added**. See also LoadRemovePart.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for LoadRemovePart l_r_p:

```
var key = l_r_p.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the LoadRemovePart. **Note that a carriage return is not added**. See also LoadRemovePart.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for LoadRemovePart l_r_p:

```
var cards = l_r_p.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last load remove_part in the model.

Arguments

- **Model** (Model)

Model to get last load remove_part in

Returns

LoadRemovePart object (or null if there are no load remove_parts in the model).

Return type

LoadRemovePart

Example

To get the last load remove_part in model m:

```
var l_r_p = LoadRemovePart.Last(m);
```

Next()

Description

Returns the next load remove_part in the model.

Arguments

No arguments

Returns

LoadRemovePart object (or null if there are no more load remove_parts in the model).

Return type

LoadRemovePart

Example

To get the load remove_part in model m after load remove_part l_r_p:

```
var l_r_p = l_r_p.Next();
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a load remove_part.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only load remove_parts from that model can be picked. If the argument is a Flag then only load remove_parts that are flagged with *limit* can be selected. If omitted, or null, any load remove_parts from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

LoadRemovePart object (or null if not picked)

Return type

LoadRemovePart

Example

To pick a load remove_part from model m giving the prompt 'Pick load remove_part from screen':

```
var l_r_p = LoadRemovePart.Pick('Pick load remove_part from screen', m);
```

Previous()

Description

Returns the previous load remove_part in the model.

Arguments

No arguments

Returns

LoadRemovePart object (or null if there are no more load remove_parts in the model).

Return type

LoadRemovePart

Example

To get the load remove_part in model m before load remove_part l_r_p:

```
var l_r_p = l_r_p.Previous();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select load remove_parts using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting load remove_parts

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only load remove_parts from that model can be selected. If the argument is a Flag then only load remove_parts that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any load remove_parts can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of load remove_parts selected or null if menu cancelled

Return type

Number

Example

To select load remove_parts from model m, flagging those selected with flag f, giving the prompt 'Select load remove_parts':

```
LoadRemovePart.Select(f, 'Select load remove_parts', m);
```

To select load remove_parts, flagging those selected with flag f but limiting selection to load remove_parts flagged with flag l, giving the prompt 'Select load remove_parts':

```
LoadRemovePart.Select(f, 'Select load remove_parts', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the load remove_part.

Arguments

- **flag** (Flag)

Flag to set on the load remove_part

Returns

No return value

Example

To set flag f for load remove_part l_r_p:

```
l_r_p.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the load remove_part. The load remove_part will be sketched until you either call LoadRemovePart.Unsketch(), LoadRemovePart.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the load remove_part is sketched. If omitted redraw is true. If you want to sketch several load remove_parts and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch load remove_part l_r_p:

```
l_r_p.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged load remove_parts in the model. The load remove_parts will be sketched until you either call LoadRemovePart.Unsketch(), LoadRemovePart.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged load remove_parts will be sketched in

- **flag** (Flag)

Flag set on the load remove_parts that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the load remove_parts are sketched. If omitted redraw is true. If you want to sketch flagged load remove_parts several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all load remove_parts flagged with flag in model m:

```
LoadRemovePart.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of load remove_parts in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing load remove_parts should be counted. If false or omitted referenced but undefined load remove_parts will also be included in the total.

Returns

number of load remove_parts

Return type

Number

Example

To get the total number of load remove_parts in model m:

```
var total = LoadRemovePart.Total(m);
```

Unblank()

Description

Unblanks the load remove_part

Arguments

No arguments

Returns

No return value

Example

To unblank load remove_part l_r_p:

```
l_r_p.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the load remove_parts in the model.

Arguments

- **Model** (Model)

Model that all load remove_parts will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the load remove_parts in model m:

```
LoadRemovePart.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged load remove_parts in the model.

Arguments

- **Model** (Model)

Model that the flagged load remove_parts will be unblanked in

- **flag** (Flag)

Flag set on the load remove_parts that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the load remove_parts in model m flagged with f:

```
LoadRemovePart.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the load remove_parts in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all load remove_parts will be unset in

- **flag** (Flag)

Flag to unset on the load remove_parts

Returns

No return value

Example

To unset the flag f on all the load remove_parts in model m:

```
LoadRemovePart.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the load remove_part.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the load remove_part is unsketched. If omitted redraw is true. If you want to unsketch several load remove_parts and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch load remove_part l_r_p:

```
l_r_p.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all load remove_parts.

Arguments

- **Model** (Model)

Model that all load remove_parts will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the load remove_parts are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all load remove_parts in model m:

```
LoadRemovePart.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged load remove_parts in the model.

Arguments

- **Model** (Model)

Model that all load remove_parts will be unsketched in

- **flag** (Flag)

Flag set on the load remove_parts that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the load remove_parts are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all load remove_parts flagged with flag in model m:

```
LoadRemovePart.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with ’method chaining’ (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

LoadRemovePart object.

Return type

LoadRemovePart

Example

To check if LoadRemovePart property l_r_p.example is a parameter by using the LoadRemovePart.GetParameter() method:

```
if (l_r_p.ViewParameters().GetParameter(l_r_p.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for load remove_part with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The load remove_part object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for load remove_part l_r_p:

```
l_r_p.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for load remove_part l_r_p:

```
l_r_p.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for load remove_part. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for load remove_part l_r_p:

```
l_r_p.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this load remove_part.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for load remove_part l_r_p:

```
var xrefs = l_r_p.Xrefs();
```

---

toString()

Description

Creates a string containing the LoadRemovePart data in keyword format. Note that this contains the keyword header and the keyword cards. See also LoadRemovePart.Keyword() and LoadRemovePart.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for LoadRemovePart l_r_p in keyword format

```
var s = l_r_p.toString();
```

LoadRigidBody class

The LoadRigidBody class gives you access to define load rigidbody cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- First(Model*[Model]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

LoadRigidBody properties

Name	Type	Description

cid	integer	Coordinate system ID
dof	integer	Applicable degrees-of-freedom
exists (read only)	logical	true if load rigidbody exists, false if referred to but not defined.
include	integer	The Include file number that the load rigidbody is in.
lcid	integer	Curve ID
m1	integer	Node 1 ID
m2	integer	Node 2 ID
m3	integer	Node 3 ID
model (read only)	integer	The Model number that the load rigidbody is in.
pid	integer	Part ID
sf	real	Curve scale factor

Detailed Description

The LoadRigidBody class allows you to create, modify, edit and manipulate load rigidbody cards. See the documentation below for more details.

Constructor

new LoadRigidBody(Model*[Model]*, pid*[integer]*, dof*[integer]*, lcid*[integer]*, sf (optional)*[real]*, cid (optional)*[integer]*, m1 (optional)*[integer]*, m2 (optional)*[integer]*, m3 (optional)*[integer]*)

Description

Create a new LoadRigidBody object.

Arguments

- **Model** (Model)

Model that load rigidbody will be created in

- **pid** (integer)

Part ID

- **dof** (integer)

Applicable degrees-of-freedom

- **lcid** (integer)

Curve ID

- **sf (optional)** (real)

Curve scale factor

- **cid (optional)** (integer)

Coordinate system ID

- **m1 (optional)** (integer)

Node 1 ID

- **m2 (optional)** (integer)

Node 2 ID

- **m3 (optional)** (integer)

Node 3 ID

Returns

LoadRigidBody object

Return type

LoadRigidBody

Example

To create a new load rigidbody in model m, for part 100, with loadcurve 9 and a scale factor of 0.5

```
var lrb = new LoadRigidBody(m, 100, 2, 9, 0.5);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a load rigidbody.

Arguments

- **Comment** (Comment)

Comment that will be attached to the load rigidbody

Returns

No return value

Example

To associate comment c to the load rigidbody lrb:

```
lrb.AssociateComment(c);
```

Blank()

Description

Blanks the load rigidbody

Arguments

No arguments

Returns

No return value

Example

To blank load rigidbody lrb:

```
lrb.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the load rigidbodies in the model.

Arguments

- **Model** (Model)

Model that all load rigidbodies will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the load rigidbodies in model m:

```
LoadRigidBody.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged load rigidbodies in the model.

Arguments

- **Model** (Model)

Model that all the flagged load rigidbodies will be blanked in

- **flag** (Flag)

Flag set on the load rigidbodies that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the load rigidbodies in model m flagged with f:

```
LoadRigidBody.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the load rigidbody is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if load rigidbody lrb is blanked:

```
if (lrb.Blanked() ) do_something...
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the load rigidbody.

Arguments

- **flag** (Flag)

Flag to clear on the load rigidbody

Returns

No return value

Example

To clear flag f for load rigidbody lrb:

```
lrb.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the load rigidbody. The target include of the copied load rigidbody can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

LoadRigidBody object

Return type

LoadRigidBody

Example

To copy load rigidbody lrb into load rigidbody z:

```
var z = lrb.Copy();
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a load rigidbody.

Arguments

- **Comment** (Comment)

Comment that will be detached from the load rigidbody

Returns

No return value

Example

To detach comment c from the load rigidbody lrb:

```
lrb.DetachComment(c);
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for load rigidbody with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The load rigidbody object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for load rigidbody lrb:

```
lrb.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for load rigidbody lrb:

```
lrb.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for load rigidbody. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for load rigidbody lrb:

```
lrb.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first load rigidbody in the model.

Arguments

- **Model** (Model)

Model to get first load rigidbody in

Returns

LoadRigidBody object (or null if there are no load rigidbodies in the model).

Return type

LoadRigidBody

Example

To get the first load rigidbody in model m:

```
var lrb = LoadRigidBody.First(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the load rigidbodies in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all load rigidbodies will be flagged in

- **flag** (Flag)

Flag to set on the load rigidbodies

Returns

No return value

Example

To flag all of the load rigidbodies with flag f in model m:

```
LoadRigidBody.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the load rigidbody is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the load rigidbody

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if load rigidbody lrb has flag f set on it:

```
if (lrb.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each load rigidbody in the model.
Note that ForEach has been designed to make looping over load rigidbodies as fast as possible and so has some limitations.
Firstly, a single temporary LoadRigidBody object is created and on each function call it is updated with the current load rigidbody data. This means that you should not try to store the LoadRigidBody object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new load rigidbodies inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all load rigidbodies are in

- **func** (function)

Function to call for each load rigidbody

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the load rigidbodies in model m:

```
LoadRigidBody.ForEach(m, test);
function test(lrb)
{
// lrb is LoadRigidBody object
}
```

To call function test for all of the load rigidbodies in model m with optional object:

```
var data = { x:0, y:0 };
LoadRigidBody.ForEach(m, test, data);
function test(lrb, extra)
{
// lrb is LoadRigidBody object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of LoadRigidBody objects or properties for all of the load rigidbodies in a model in PRIMER. If the optional property argument is not given then an array of LoadRigidBody objects is returned. If the property argument is given, that property value for each load rigidbody is returned in the array instead of a LoadRigidBody object

Arguments

- **Model** (Model)

Model to get load rigidbodies from

- **property (optional)** (string)

Name for property to get for all load rigidbodies in the model

Returns

Array of LoadRigidBody objects or properties

Return type

Array

Example

To make an array of LoadRigidBody objects for all of the load rigidbodies in model m:

```
var a = LoadRigidBody.GetAll(m);
```

To return an array containing the value of property ’foo’ (for example ’x’ for a node) for each load rigidbody in model m:

```
var a = LoadRigidBody.GetAll(m, ’foo’);
```

GetComments()

Description

Extracts the comments associated to a load rigidbody.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the load rigidbody lrb:

```
var comm_array = lrb.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of LoadRigidBody objects for all of the flagged load rigidbodies in a model in PRIMER If the optional property argument is not given then an array of LoadRigidBody objects is returned. If the property argument is given, then that property value for each load rigidbody is returned in the array instead of a LoadRigidBody object

Arguments

- **Model** (Model)

Model to get load rigidbodies from

- **flag** (Flag)

Flag set on the load rigidbodies that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged load rigidbodies in the model

Returns

Array of LoadRigidBody objects or properties

Return type

Array

Example

To make an array of LoadRigidBody objects for all of the load rigidbodies in model m flagged with f:

```
var lrb = LoadRigidBody.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the load rigidbodies in model m flagged with f:

```
var a = LoadRigidBody.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the LoadRigidBody object for a load rigidbody ID.

Arguments

- **Model** (Model)

Model to find the load rigidbody in

- **number** (integer)

number of the load rigidbody you want the LoadRigidBody object for

Returns

LoadRigidBody object (or null if load rigidbody does not exist).

Return type

LoadRigidBody

Example

To get the LoadRigidBody object for load rigidbody 100 in model m

```
var lrb = LoadRigidBody.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a LoadRigidBody property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the LoadRigidBody.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

load rigidbody property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if LoadRigidBody property lrb.example is a parameter:

```
Options.property_parameter_names = true;
if (lrb.GetParameter(lrb.example) ) do_something...
Options.property_parameter_names = false;
```

To check if LoadRigidBody property lrb.example is a parameter by using the GetParameter method:

```
if (lrb.ViewParameters().GetParameter(lrb.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this load rigidbody (*LOAD_RIGIDBODY). **Note that a carriage return is not added**. See also LoadRigidBody.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for load rigidbody lrb:

```
var key = lrb.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the load rigidbody. **Note that a carriage return is not added**. See also LoadRigidBody.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for load rigidbody lrb:

```
var cards = lrb.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last load rigidbody in the model.

Arguments

- **Model** (Model)

Model to get last load rigidbody in

Returns

LoadRigidBody object (or null if there are no load rigidbodies in the model).

Return type

LoadRigidBody

Example

To get the last load rigidbody in model m:

```
var lrb = LoadRigidBody.Last(m);
```

Next()

Description

Returns the next load rigidbody in the model.

Arguments

No arguments

Returns

LoadRigidBody object (or null if there are no more load rigidbodies in the model).

Return type

LoadRigidBody

Example

To get the load rigidbody in model m after load rigidbody lrb:

```
var lrb = lrb.Next();
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a load rigidbody.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only load rigidbodies from that model can be picked. If the argument is a Flag then only load rigidbodies that are flagged with *limit* can be selected. If omitted, or null, any load rigidbodies from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

LoadRigidBody object (or null if not picked)

Return type

LoadRigidBody

Example

To pick a load rigidbody from model m giving the prompt 'Pick load rigidbody from screen':

```
var lrb = LoadRigidBody.Pick('Pick load rigidbody from screen', m);
```

Previous()

Description

Returns the previous load rigidbody in the model.

Arguments

No arguments

Returns

LoadRigidBody object (or null if there are no more load rigidbodies in the model).

Return type

LoadRigidBody

Example

To get the load rigidbody in model m before load rigidbody lrb:

```
var lrb = lrb.Previous();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select load rigidbodies using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting load rigidbodies

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only load rigidbodies from that model can be selected. If the argument is a Flag then only load rigidbodies that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any load rigidbodies can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of load rigidbodies selected or null if menu cancelled

Return type

Number

Example

To select load rigidbodies from model m, flagging those selected with flag f, giving the prompt 'Select load rigidbodies':

```
LoadRigidBody.Select(f, 'Select load rigidbodies', m);
```

To select load rigidbodies, flagging those selected with flag f but limiting selection to load rigidbodies flagged with flag l, giving the prompt 'Select load rigidbodies':

```
LoadRigidBody.Select(f, 'Select load rigidbodies', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the load rigidbody.

Arguments

- **flag** (Flag)

Flag to set on the load rigidbody

Returns

No return value

Example

To set flag f for load rigidbody lrb:

```
lrb.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the load rigidbody. The load rigidbody will be sketched until you either call LoadRigidBody.Unsketch(), LoadRigidBody.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the load rigidbody is sketched. If omitted redraw is true. If you want to sketch several load rigidbodies and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch load rigidbody lrb:

```
lrb.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged load rigidbodies in the model. The load rigidbodies will be sketched until you either call LoadRigidBody.Unsketch(), LoadRigidBody.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged load rigidbodies will be sketched in

- **flag** (Flag)

Flag set on the load rigidbodies that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the load rigidbodies are sketched. If omitted redraw is true. If you want to sketch flagged load rigidbodies several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all load rigidbodies flagged with flag in model m:

```
LoadRigidBody.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of load rigidbodies in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing load rigidbodies should be counted. If false or omitted referenced but undefined load rigidbodies will also be included in the total.

Returns

number of load rigidbodies

Return type

Number

Example

To get the total number of load rigidbodies in model m:

```
var total = LoadRigidBody.Total(m);
```

Unblank()

Description

Unblanks the load rigidbody

Arguments

No arguments

Returns

No return value

Example

To unblank load rigidbody lrb:

```
lrb.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the load rigidbodies in the model.

Arguments

- **Model** (Model)

Model that all load rigidbodies will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the load rigidbodies in model m:

```
LoadRigidBody.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged load rigidbodies in the model.

Arguments

- **Model** (Model)

Model that the flagged load rigidbodies will be unblanked in

- **flag** (Flag)

Flag set on the load rigidbodies that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the load rigidbodies in model m flagged with f:

```
LoadRigidBody.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the load rigidbodies in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all load rigidbodies will be unset in

- **flag** (Flag)

Flag to unset on the load rigidbodies

Returns

No return value

Example

To unset the flag f on all the load rigidbodies in model m:

```
LoadRigidBody.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the load rigidbody.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the load rigidbody is unsketched. If omitted redraw is true. If you want to unsketch several load rigidbodies and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch load rigidbody lrb:

```
lrb.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all load rigidbodies.

Arguments

- **Model** (Model)

Model that all load rigidbodies will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the load rigidbodies are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all load rigidbodies in model m:

```
LoadRigidBody.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged load rigidbodies in the model.

Arguments

- **Model** (Model)

Model that all load rigidbodies will be unsketched in

- **flag** (Flag)

Flag set on the load rigidbodies that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the load rigidbodies are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all load rigidbodies flagged with flag in model m:

```
LoadRigidBody.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

LoadRigidBody object.

Return type

LoadRigidBody

Example

To check if LoadRigidBody property lrb.example is a parameter by using the LoadRigidBody.GetParameter() method:

```
if (lrb.ViewParameters().GetParameter(lrb.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for load rigidbody with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description

autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The load rigidbody object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for load rigidbody lrb:

```
lrb.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for load rigidbody lrb:

```
lrb.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for load rigidbody. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for load rigidbody lrb:

```
lrb.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this load rigidbody.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for load rigidbody lrb:

```
var xrefs = lrb.Xrefs();
```

---

toString()

Description

Creates a string containing the load rigidbody data in keyword format. Note that this contains the keyword header and the keyword cards. See also LoadRigidBody.Keyword() and LoadRigidBody.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for load rigidbody lrb in keyword format

```
var s = lrb.toString();
```

LoadShell class

The LoadShell class gives you access to define *LOAD_SHELL cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

LoadShell constants

Name	Description
LoadShell.ELEMENT	Load is *LOAD_SHELL_ELEMENT.
LoadShell.SET	LOAD is *LOAD_SHELL_SET.

LoadShell properties

Name	Type	Description
at	real	Arrival time for pressure
eid	integer	Shell ID or shell set ID
exists (read only)	logical	true if load shell exists, false if referred to but not defined.
heading	string	LoadShell heading
id	logical	true if _ID option is set, false if not
include	integer	The Include file number that the load shell is in.
label	integer	LoadShell number.
lcid	integer	Curve ID
lsid	integer	LoadShell number (identical to label).
model (read only)	integer	The Model number that the load shell is in.
sf	real	Curve scale factor
type	constant	The Load Node type. Can be LoadShell.ELEMENT or LoadShell.SET.

Detailed Description

The LoadShell class allows you to create, modify, edit and manipulate *LOAD_SHELL cards. See the documentation below for more details.

Constructor

new LoadShell(Model*[Model]*, type*[constant]*, eid*[integer]*, lcid*[integer]*, sf (optional)*[real]*, at (optional)*[real]*, lsid (optional)*[integer]*, heading (optional)*[string]*)

Description

Create a new LoadShell object.

Arguments

- **Model** (Model)

Model that LoadShell will be created in

- **type** (constant)

Specify the type of LoadShell (Can be LoadShell.ELEMENT or LoadShell.SET)

- **eid** (integer)

Shell ID or shell set ID

- **lcid** (integer)

Curve ID

- **sf (optional)** (real)

Curve scale factor

- **at (optional)** (real)

Arrival time for pressure

- **lsid (optional)** (integer)

LoadShell number

- **heading (optional)** (string)

Title for the LoadShell

Returns

LoadShell object

Return type

LoadShell

Example

To create a new load shell in model m, of type SET, with loadcurve 9 and a scale factor of 0.5

```
var b = new LoadShell(m, LoadShell.SET, 100, 2, 9, 0.5);
```

Details of functions

AssociateComment(Comment[*Comment*])

Description

Associates a comment with a load shell.

Arguments

- **Comment** (Comment)

Comment that will be attached to the load shell

Returns

No return value

Example

To associate comment c to the load shell ls:

```
ls.AssociateComment(c);
```

Blank()

Description

Blanks the load shell

Arguments

No arguments

Returns

No return value

Example

To blank load shell ls:

```
ls.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the load shells in the model.

Arguments

- **Model** (Model)

Model that all load shells will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the load shells in model m:

```
LoadShell.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged load shells in the model.

Arguments

- **Model** (Model)

Model that all the flagged load shells will be blanked in

- **flag** (Flag)

Flag set on the load shells that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the load shells in model m flagged with f:

```
LoadShell.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the load shell is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if load shell ls is blanked:

```
if (ls.Blanked() ) do_something...
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the load shell.

Arguments

- **flag** (Flag)

Flag to clear on the load shell

Returns

No return value

Example

To clear flag f for load shell ls:

```
ls.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the load shell. The target include of the copied load shell can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

LoadShell object

Return type

LoadShell

Example

To copy load shell ls into load shell z:

```
var z = ls.Copy();
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a load shell.

Arguments

- **Comment** (Comment)

Comment that will be detached from the load shell

Returns

No return value

Example

To detach comment c from the load shell ls:

```
ls.DetachComment(c);
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for load shell with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The load shell object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for load shell ls:

```
ls.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for load shell ls:

```
ls.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for load shell. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for load shell ls:

```
ls.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first load shell in the model.

Arguments

- **Model** (Model)

Model to get first load shell in

Returns

LoadShell object (or null if there are no load shells in the model).

Return type

LoadShell

Example

To get the first load shell in model m:

```
var ls = LoadShell.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free load shell label in the model. Also see LoadShell.LastFreeLabel(), LoadShell.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free load shell label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

LoadShell label.

Return type

Number

Example

To get the first free load shell label in model m:

```
var label = LoadShell.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the load shells in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all load shells will be flagged in

- **flag** (Flag)

Flag to set on the load shells

Returns

No return value

Example

To flag all of the load shells with flag f in model m:

```
LoadShell.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the load shell is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the load shell

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if load shell ls has flag f set on it:

```
if (ls.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each load shell in the model.
Note that ForEach has been designed to make looping over load shells as fast as possible and so has some limitations.
Firstly, a single temporary LoadShell object is created and on each function call it is updated with the current load shell data. This means that you should not try to store the LoadShell object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new load shells inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all load shells are in

- **func** (function)

Function to call for each load shell

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the load shells in model m:

```
LoadShell.ForEach(m, test);
function test(ls)
{
// ls is LoadShell object
}
```

To call function test for all of the load shells in model m with optional object:

```
var data = { x:0, y:0 };
LoadShell.ForEach(m, test, data);
function test(ls, extra)
{
// ls is LoadShell object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of LoadShell objects or properties for all of the load shells in a model in PRIMER. If the optional property argument is not given then an array of LoadShell objects is returned. If the property argument is given, that property value for each load shell is returned in the array instead of a LoadShell object

Arguments

- **Model** (Model)

Model to get load shells from

- **property (optional)** (string)

Name for property to get for all load shells in the model

Returns

Array of LoadShell objects or properties

Return type

Array

Example

To make an array of LoadShell objects for all of the load shells in model m:

```
var a = LoadShell.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each load shell in model m:

```
var a = LoadShell.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a load shell.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the load shell ls:

```
var comm_array = ls.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of LoadShell objects for all of the flagged load shells in a model in PRIMER If the optional property argument is not given then an array of LoadShell objects is returned. If the property argument is given, then that property value for each load shell is returned in the array instead of a LoadShell object

Arguments

- **Model** (Model)

Model to get load shells from

- **flag** (Flag)

Flag set on the load shells that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged load shells in the model

Returns

Array of LoadShell objects or properties

Return type

Array

Example

To make an array of LoadShell objects for all of the load shells in model m flagged with f:

```
var ls = LoadShell.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the load shells in model m flagged with f:

```
var a = LoadShell.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the LoadShell object for a load shell ID.

Arguments

- **Model** (Model)

Model to find the load shell in

- **number** (integer)

number of the load shell you want the LoadShell object for

Returns

LoadShell object (or null if load shell does not exist).

Return type

LoadShell

Example

To get the LoadShell object for load shell 100 in model m

```
var ls = LoadShell.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a LoadShell property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the LoadShell.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

load shell property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if LoadShell property ls.example is a parameter:

```
Options.property_parameter_names = true;
if (ls.GetParameter(ls.example) ) do_something...
Options.property_parameter_names = false;
```

To check if LoadShell property ls.example is a parameter by using the GetParameter method:

```
if (ls.ViewParameters().GetParameter(ls.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this LoadShell (*LOAD_SHELL_xxxx). **Note that a carriage return is not added**. See also LoadShell.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for LoadShell m:

```
var key = m.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the LoadShell. **Note that a carriage return is not added**. See also LoadShell.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for LoadShell l:

```
var cards = l.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last load shell in the model.

Arguments

- **Model** (Model)

Model to get last load shell in

Returns

LoadShell object (or null if there are no load shells in the model).

Return type

LoadShell

Example

To get the last load shell in model m:

```
var ls = LoadShell.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free load shell label in the model. Also see LoadShell.FirstFreeLabel(), LoadShell.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free load shell label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

LoadShell label.

Return type

Number

Example

To get the last free load shell label in model m:

```
var label = LoadShell.LastFreeLabel(m);
```

Next()

Description

Returns the next load shell in the model.

Arguments

No arguments

Returns

LoadShell object (or null if there are no more load shells in the model).

Return type

LoadShell

Example

To get the load shell in model m after load shell ls:

```
var ls = ls.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) load shell label in the model. Also see LoadShell.FirstFreeLabel(), LoadShell.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free load shell label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

LoadShell label.

Return type

Number

Example

To get the next free load shell label in model m:

```
var label = LoadShell.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a load shell.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only load shells from that model can be picked. If the argument is a Flag then only load shells that are flagged with *limit* can be selected. If omitted, or null, any load shells from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

LoadShell object (or null if not picked)

Return type

LoadShell

Example

To pick a load shell from model m giving the prompt 'Pick load shell from screen':

```
var ls = LoadShell.Pick('Pick load shell from screen', m);
```

Previous()

Description

Returns the previous load shell in the model.

Arguments

No arguments

Returns

LoadShell object (or null if there are no more load shells in the model).

Return type

LoadShell

Example

To get the load shell in model m before load shell ls:

```
var ls = ls.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the load shells in the model.

Arguments

- **Model** (Model)

Model that all load shells will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the load shells in model m, from 1000000:

```
LoadShell.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged load shells in the model.

Arguments

- **Model** (Model)

Model that all the flagged load shells will be renumbered in

- **flag** (Flag)

Flag set on the load shells that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the load shells in model m flagged with f, from 1000000:

```
LoadShell.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select load shells using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting load shells

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only load shells from that model can be selected. If the argument is a Flag then only load shells that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any load shells can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of load shells selected or null if menu cancelled

Return type

Number

Example

To select load shells from model m, flagging those selected with flag f, giving the prompt 'Select load shells':

```
LoadShell.Select(f, 'Select load shells', m);
```

To select load shells, flagging those selected with flag f but limiting selection to load shells flagged with flag l, giving the prompt 'Select load shells':

```
LoadShell.Select(f, 'Select load shells', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the load shell.

Arguments

- **flag** (Flag)

Flag to set on the load shell

Returns

No return value

Example

To set flag f for load shell ls:

```
ls.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the load shell. The load shell will be sketched until you either call LoadShell.Unsketch(), LoadShell.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the load shell is sketched. If omitted redraw is true. If you want to sketch several load shells and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch load shell ls:

```
ls.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged load shells in the model. The load shells will be sketched until you either call LoadShell.Unsketch(), LoadShell.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged load shells will be sketched in

- **flag** (Flag)

Flag set on the load shells that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the load shells are sketched. If omitted redraw is true. If you want to sketch flagged load shells several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all load shells flagged with flag in model m:

```
LoadShell.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of load shells in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing load shells should be counted. If false or omitted referenced but undefined load shells will also be included in the total.

Returns

number of load shells

Return type

Number

Example

To get the total number of load shells in model m:

```
var total = LoadShell.Total(m);
```

Unblank()

Description

Unblanks the load shell

Arguments

No arguments

Returns

No return value

Example

To unblank load shell ls:

```
ls.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the load shells in the model.

Arguments

- **Model** (Model)

Model that all load shells will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the load shells in model m:

```
LoadShell.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged load shells in the model.

Arguments

- **Model** (Model)

Model that the flagged load shells will be unblanked in

- **flag** (Flag)

Flag set on the load shells that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the load shells in model m flagged with f:

```
LoadShell.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the load shells in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all load shells will be unset in

- **flag** (Flag)

Flag to unset on the load shells

Returns

No return value

Example

To unset the flag f on all the load shells in model m:

```
LoadShell.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the load shell.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the load shell is unsketched. If omitted redraw is true. If you want to unsketch several load shells and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch load shell ls:

```
ls.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all load shells.

Arguments

- **Model** (Model)

Model that all load shells will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the load shells are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all load shells in model m:

```
LoadShell.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged load shells in the model.

Arguments

- **Model** (Model)

Model that all load shells will be unsketched in

- **flag** (Flag)

Flag set on the load shells that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the load shells are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all load shells flagged with flag in model m:

```
LoadShell.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

LoadShell object.

Return type

LoadShell

Example

To check if LoadShell property ls.example is a parameter by using the LoadShell.GetParameter() method:

```
if (ls.ViewParameters().GetParameter(ls.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for load shell with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The load shell object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for load shell ls:

```
ls.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for load shell ls:

```
ls.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for load shell. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for load shell ls:

```
ls.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this load shell.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for load shell ls:

```
var xrefs = ls.Xrefs();
```

---

toString()

Description

Creates a string containing the LoadShell data in keyword format. Note that this contains the keyword header and the keyword cards. See also LoadShell.Keyword() and LoadShell.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for LoadShell l in keyword format

```
var s = l.toString();
```

Material class

The Material class gives you access to material cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AddOptionalCards()
- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- Density()
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- ExtractColour()
- Flagged(flag*[Flag]*)
- GetAddDamageGissmoData()
- GetAddErosionData()
- GetComments()
- GetParameter(prop*[string]*)
- GetPropertyByIndex(index*[integer]*)
- GetPropertyByName(acronym*[string]*)
- GetPropertyByRowCol(row*[integer]*, col*[integer]*)
- GetPropertyNameForIndex(index*[integer]*)
- GetPropertyNameForRowCol(row*[integer]*, col*[integer]*)
- Keyword(index (optional)*[integer]*)

- KeywordCards(index (optional)*[integer]*)
- Next()
- PoissonsRatio()
- Previous()
- SetAddDamageGissmoData(data*[object]*)
- SetAddErosionData(data*[object]*)
- SetFlag(flag*[Flag]*)
- SetPropertyByIndex(index*[integer]*, value*[integer/real for numeric properties, string for character properties]*)
- SetPropertyByName(acronym*[string]*, value*[integer/real for numeric properties, string for character properties]*)
- SetPropertyByRowCol(row*[integer]*, col*[integer]*, value*[integer/real for numeric properties, string for character properties]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- YieldStress()
- YoungsModulus()
- toString()

Material properties

Name	Type	Description
addDamageGissmo	logical	True if *MAT_ADD_DAMAGE_GISSMO exists for material, false if not defined
addErosion	logical	True if *MAT_ADD_EROSION exists for material, false if not defined
addKeywords (read only)	integer	The number of **extra** *MAT_ADD_xxxx keywords that this material definition has. Note that if there is only a single *MAT_ADD_xxxx keyword for an ID this will be 0. For example, if for material 1 both a *MAT_PIECEWISE_LINEAR_PLASTICITY card and a *MAT_ADD_EROSION card exist then this will return 1. If for material 2 only a *MAT_ADD_EROSION card exists then this will return 0. Also see Material.Keyword() and Material.KeywordCards()
colour	Colour	The colour of the material
cols (read only)	real	The number of columns of data the material has
exists (read only)	logical	true if material exists, false if referred to but not defined.
include	integer	The Include file number that the material is in.
label	integer or string	Material number or character label. Also see the mid property which is an alternative name for this.
mid	integer or string	Material number or character label. Also see the label property which is an alternative name for this.
model (read only)	integer	The Model number that the material is in.
optionalCards (read only)	integer	The number of optional extra cards that this material definition can have. Also see Material.AddOptionalCards()
properties	integer	The total number of properties that the material has
rows (read only)	integer	The number of rows of data the material has
title	string	Material title
transparency	integer	The transparency of the material (0-100) 0% is opaque, 100% is transparent.
type	string	The material type name(e.g. 'ELASTIC', 'RIGID' etc).
typeNumber	string	The material type number (e.g. '001', '034M').

Detailed Description

The Material class allows you to create, modify, edit and manipulate material cards. See the documentation below for more details.

Constructor

new Material(Model*[Model]*, mid*[integer or string]*, type*[string]*)

Description

Create a new Material object.

Arguments

- **Model** (Model)

Model that material will be created in

- **mid** (integer or string)

Material number or character label

- **type** (string)

Material type. Either give the Ansys LS-DYNA material name or 3 digit number.

Returns

Material object

Return type

Material

Example

To create a new rigid material in model m with label 100

```
var mat = new Material(m, 100, "RIGID");
```

or

```
var mat = new Material(m, 100, "020");
```

or

```
var mat = new Material(m, 100, "*MAT_RIGID");
```

or

```
var mat = new Material(m, 100, "*MAT_020");
```

Details of functions

AddOptionalCards()

Description

Adds any optional cards for the material.
Some materials have extra optional cards in the input. If they are there Ansys LS-DYNA will read them but they are not required input. For example a material could have three required cards and one extra optional card. If PRIMER reads this material from a keyword file and it only has the three required cards then the properties in the material will only be defined for those cards. i.e. there will not be any properties in the material for the extra optional line.
If you edit the material interactively in PRIMER then the extra optional card will be shown so you can add values if required. When writing the material to a keyword file the extra optional card will be omitted if none of the fields are used.
If you want to add one of the properties for the extra optional card in JavaScript this method will ensure that the extra card is defined and the properties added to the material as zero values. You can then use Material.SetPropertyByIndex(), Material.SetPropertyByName() or Material.SetPropertyByRowCol() as normal to set the properties. Also see the optionalCards property.

Arguments

No arguments

Returns

no return value

Example

To add any optional cards for material m:

```
m.AddOptionalCards();
```

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a material.

Arguments

- **Comment** (Comment)

Comment that will be attached to the material

Returns

No return value

Example

To associate comment c to the material m:

```
m.AssociateComment(c);
```

Blank()

Description

Blanks the material

Arguments

No arguments

Returns

No return value

Example

To blank material m:

```
m.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the materials in the model.

Arguments

- **Model** (Model)

Model that all materials will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the materials in model m:

```
Material.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged materials in the model.

Arguments

- **Model** (Model)

Model that all the flagged materials will be blanked in

- **flag** (Flag)

Flag set on the materials that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the materials in model m flagged with f:

```
Material.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the material is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if material m is blanked:

```
if (m.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse material m:

```
m.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the material.

Arguments

- **flag** (Flag)

Flag to clear on the material

Returns

No return value

Example

To clear flag f for material m:

```
m.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the material. The target include of the copied material can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

Material object

Return type

Material

Example

To copy material m into material z:

```
var z = m.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a material

Arguments

- **Model** (Model)

Model that the material will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

Material object (or null if not made).

Return type

Material

Example

To start creating a material in model m:

```
var m = Material.Create(m);
```

Density()

Description

Get the density material.

Arguments

No arguments

Returns

real

Return type

Number

Example

To get the density for material m:

```
var density = m.Density();
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a material.

Arguments

- **Comment** (Comment)

Comment that will be detached from the material

Returns

No return value

Example

To detach comment c from the material m:

```
m.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit material m:

```
m.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for material with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The material object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for material m:

```
m.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for material m:

```
m.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for material. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for material m:

```
m.Error("My custom error");
```

ExtractColour()

Description

Extracts the **actual** colour used for material.
By default in PRIMER many entities such as elements get their colour automatically from the part that they are in. PRIMER cycles through 13 default colours based on the label of the entity. In this case the material colour property will return the value Colour.PART instead of the actual colour. This method will return the actual colour which is used for drawing the material.

Arguments

No arguments

Returns

colour value (integer)

Return type

Number

Example

To return the colour used for drawing material m:

```
var colour = m.ExtractColour();
```

First(Model[*Model*]) [static]

Description

Returns the first material in the model.

Arguments

- **Model** (Model)

Model to get first material in

Returns

Material object (or null if there are no materials in the model).

Return type

Material

Example

To get the first material in model m:

```
var m = Material.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free material label in the model. Also see Material.LastFreeLabel(), Material.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free material label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

Material label.

Return type

Number

Example

To get the first free material label in model m:

```
var label = Material.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the materials in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all materials will be flagged in

- **flag** (Flag)

Flag to set on the materials

Returns

No return value

Example

To flag all of the materials with flag f in model m:

```
Material.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the material is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the material

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if material m has flag f set on it:

```
if (m.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each material in the model.
Note that ForEach has been designed to make looping over materials as fast as possible and so has some limitations.
Firstly, a single temporary Material object is created and on each function call it is updated with the current material data. This means that you should not try to store the Material object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new materials inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all materials are in

- **func** (function)

Function to call for each material

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the materials in model m:

```
Material.ForEach(m, test);
function test(m)
{
// m is Material object
}
```

To call function test for all of the materials in model m with optional object:

```
var data = { x:0, y:0 };
Material.ForEach(m, test, data);
function test(m, extra)
{
// m is Material object
// extra is data
}
```

GetAddDamageGissmoData()

Description

Returns the *MAT_ADD_DAMAGE_GISSMO data of material.

Arguments

No arguments

Returns

Object with the following properties:

Name	Type	Description
biaxf	real	Reduction factor for regularization at triaxiality=2/3
dcrit	real	Damage treshold value
dmgexp	real	Exponent for nonlinear damage accumalation
dtyp	real	Flag for GISSMO damage type
ecrit	real/integer	Critical plastic strain (Curve/ table ID if negative)
fadexp	real/integer	Exponent for damage-related stress fadeout (Curve/ table ID if negative)
hisvn	real	History variable used to evaluate th 3-D table LCSDG
instf	integer	Flag for governing the behavior of instability measure F and fading exponent FADEXP
lcdlim	integer	Curve ID: damage limit values as a function of triaxiality
lcregd	integer	Curve/ table ID (positive) or Table ID (negative): Element-size dependent fading exponent
lcsdg	integer	Curve/ table ID (positive) or Function ID (negative): Failure strain curve/table or function
lcsoft	integer	Soft reduction factor for failure strain in crashfront elements.
lcsrs	integer	Curve/ table ID: Failure strain rate scaling factor v/s strain rate
lp2bi	real	Option to use bending indicator instead of the Lode parameter
midfail	integer	Mid-plane failure option for shell elements and GISSMO
numfip	real	Number of failed integration points prior to element deletion
refsz	real	Reference element size
rgtr1	real	First triaxiality value for optional "tub-shaped" regularization
rgtr2	real	Second triaxiality value for optional "tub-shaped" regularization
shrf	real	Reduction factor for regularization at triaxiality=0
soft	real	Softening reduction factor for failure strain in crashfront elements
stochastic	logical	stochastic = true if _STOCHASTIC is ON. Otherwise, _STOCHASTIC is OFF
volfrac	real	Volume fraction required to fail before element deletion

Return type

object

Example

To get the *MAT_ADD_DAMAGE_GISSMO data of material m:

```
m.GetAddDamageGissmoData();
```

GetAddErosionData()

Description

Returns the *MAT_ADD_EROSION data of material. Note that this method does not support pre-R11 properties.

Arguments

No arguments

Returns

Object with the following properties:

Name	Type	Description
dteflt	real	Time period for the low pass filter
dtmin	real	Minimum time step size at failure
effeps	real	Maximum effective strain at failure
engcrt	real	Critical energy for nonlocal failure criterion
epssh	real	Shear strain at failure
epsthin	real	Thinning strain at failure for shells
excl	real	The exclusion number
failtm	real	Failure time
idam	integer	Flag for damage model
impulse	real	Stress impulse for failure
lceps12	integer	Load curve ID defining in-plane shear strain limit vs elem size
lceps13	integer	Load curve ID defining through-thickness shear strain limit vs elem size
lcepsmx	integer	Load curve ID defining in-plane major strain limit vs elem size
lcfld	integer	Curve (negative) or table (positive) ID: Forming limit diagram
lcregd	integer	Curve ID: Element-size dependent fading exponent
mneps	real	Minimum principal strain at failure
mnpres	real	Pressure at failure
mxeps	real/integer	Principal strain at failure (curve ID if negative)
mxpres	real	Maximum pressure at failure
mxtmp	real	Maximum temperature at failure
ncs	real	Number of failure conditions to satisfy before failure occurs
nsff	real	Number of explicit time step cycles for stress fade-out used in the LCFLD criterion
numfip	real	Number of failed integration points prior to element deletion
radcrt	real	Critical radius for nonlocal failure criterion
sigp1	real	Principal stress at failure
sigth	real	Threshold stress
sigvm	real/integer	Equivalent stress at failure (curve ID if negative)
voleps	real	Volumetric strain at failure
volfrac	real	The volume fraction required to fail before the element is deleted

Return type

object

Example

To get the *MAT_ADD_EROSION data of material m:

```
m.GetAddErosionData();
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Material objects or properties for all of the materials in a model in PRIMER. If the optional property argument is not given then an array of Material objects is returned. If the property argument is given, that property value for each material is returned in the array instead of a Material object

Arguments

- **Model** (Model)

Model to get materials from

- **property (optional)** (string)

Name for property to get for all materials in the model

Returns

Array of Material objects or properties

Return type

Array

Example

To make an array of Material objects for all of the materials in model m:

```
var a = Material.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each material in model m:

```
var a = Material.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a material.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the material m:

```
var comm_array = m.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of Material objects for all of the flagged materials in a model in PRIMER If the optional property argument is not given then an array of Material objects is returned. If the property argument is given, then that property value for each material is returned in the array instead of a Material object

Arguments

- **Model** (Model)

Model to get materials from

- **flag** (Flag)

Flag set on the materials that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged materials in the model

Returns

Array of Material objects or properties

Return type

Array

Example

To make an array of Material objects for all of the materials in model m flagged with f:

```
var m = Material.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the materials in model m flagged with f:

```
var a = Material.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the Material object for a material ID.

Arguments

- **Model** (Model)

Model to find the material in

- **number** (integer)

number of the material you want the Material object for

Returns

Material object (or null if material does not exist).

Return type

Material

Example

To get the Material object for material 100 in model m

```
var m = Material.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a Material property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the Material.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

material property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if Material property m.example is a parameter:

```
Options.property_parameter_names = true;
if (m.GetParameter(m.example) ) do_something...
Options.property_parameter_names = false;
```

To check if Material property m.example is a parameter by using the GetParameter method:

```
if (m.ViewParameters().GetParameter(m.example) ) do_something...
```

GetPropertyByIndex(index*[integer]*)

Description

Returns the value of property at index *index* for this Material object or null if no property exists.

Arguments

- **index** (integer)

The index of the property value to retrieve. (the number of properties can be found from properties) **Note that indices start at 0**. There is no link between indices and rows/columns so adjacent fields on a line for a material may not have adjacent indices.

Returns

Property value (real/integer)

Return type

Number

Example

To return the property at index 2, for material m:

```
var prop = m.GetPropertyByIndex(2);
```

GetPropertyByName(acronym*[string]*)

Description

Returns the value of property string *acronym* for this Material object or null if no property exists.

Arguments

- **acronym** (string)

The acronym of the property value to retrieve

Returns

Property value (real/integer)

Return type

Number

Example

To return the value of RO for material m:

```
var ro = m.GetPropertyByName("RO");
```

GetPropertyByRowCol(row*[integer]*, col*[integer]*)

Description

Returns the value of the property for row and col for this Material object or null if no property exists. **Note that rows and columns start at 0**.

Arguments

- **row** (integer)

The row of the property value to retrieve

- **col** (integer)

The column of the property value to retrieve

Returns

Property value (real/integer)

Return type

Number

Example

To return the value of the property at row 0, column 1 for material m:

```
var prop = m.GetPropertyByRowCol(0, 1);
```

GetPropertyNameForIndex(index*[integer]*)

Description

Returns the name of the property at index *index* for this Material object or null if there is no property.

Arguments

- **index** (integer)

The index of the property name to retrieve. (the number of properties can be found from properties) **Note that indices start at 0**. There is no link between indices and rows/columns so adjacent fields on a line for a material may not have

adjacent indices.

Returns

Property name (string)

Return type

String

Example

To return the name of the property at index 2, for material m:

```
var name = m.GetPropertyNameForIndex(2);
```

GetPropertyNameForRowCol(row*[integer]*, col*[integer]*)

Description

Returns the name of the property at row and col for this Material object or null if there is no property. **Note that rows and columns start at 0**.

Arguments

- **row** (integer)

The row of the property name to retrieve

- **col** (integer)

The column of the property name to retrieve

Returns

Property name (string)

Return type

String

Example

To return the name of the property at row 0, column 1 for material m:

```
var name = m.GetPropertyNameForRowCol(0, 1);
```

Keyword(index (optional)*[integer]*)

Description

Returns the keyword for this material (e.g. *MAT_RIGID, *MAT_ELASTIC etc). **Note that a carriage return is not added**. See also Material.KeywordCards()

Arguments

- **index (optional)** (integer)

If this argument is not given then the material keyword is returned as normal. However if the material also has *MAT_ADD_xxxx cards defined for it (e.g. *MAT_ADD_EROSION) then the index can be used to return the title for the *MAT_ADD card instead. The index value starts from zero. The number of *MAT_ADD cards can be found from the addKeywords property

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for material m:

```
var key = m.Keyword();
```

To print all of the keywords and keyword cards for any *MAT_ADD cards for material m:

```
for (i=0; i<m.addKeywords; i++)
{
    Message(m.Keyword(i));
    Message(m.KeywordCards(i));
}
```

KeywordCards(index (optional)*[integer]*)

Description

Returns the keyword cards for the material. **Note that a carriage return is not added**. See also Material.Keyword()

Arguments

- **index (optional)** (integer)

If this argument is not given then the material keyword cards are returned as normal. However if the material also has *MAT_ADD_xxxx cards defined for it (e.g. *MAT_ADD_EROSION) then the index can be used to return the cards for the *MAT_ADD card instead. The index value starts from zero. The number of *MAT_ADD cards can be found from the addKeywords property

Returns

string containing the cards.

Return type

String

Example

To get the cards for material m:

```
var cards = m.KeywordCards();
```

To print all of the keywords and keyword cards for any *MAT_ADD cards for material m:

```
for (i=0; i<m.addKeywords; i++)
{
    Message(m.Keyword(i));
    Message(m.KeywordCards(i));
}
```

Last(Model*[Model]*) [static]

Description

Returns the last material in the model.

Arguments

- **Model** (Model)

Model to get last material in

Returns

Material object (or null if there are no materials in the model).

Return type

Material

Example

To get the last material in model m:

```
var m = Material.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free material label in the model. Also see Material.FirstFreeLabel(), Material.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free material label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

Material label.

Return type

Number

Example

To get the last free material label in model m:

```
var label = Material.LastFreeLabel(m);
```

Next()

Description

Returns the next material in the model.

Arguments

No arguments

Returns

Material object (or null if there are no more materials in the model).

Return type

Material

Example

To get the material in model m after material m:

```
var m = m.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) material label in the model. Also see Material.FirstFreeLabel(), Material.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free material label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

Material label.

Return type

Number

Example

To get the next free material label in model m:

```
var label = Material.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a material.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only materials from that model can be picked. If the argument is a Flag then only materials that are flagged with *limit* can be selected. If omitted, or null, any materials from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

Material object (or null if not picked)

Return type

Material

Example

To pick a material from model m giving the prompt 'Pick material from screen':

```
var m = Material.Pick('Pick material from screen', m);
```

PoissonsRatio()

Description

Get Poissons ratio for the material.

Arguments

No arguments

Returns

real

Return type

Number

Example

To get Poissons ratio for material m:

```
var pr = m.PoissonsRatio(f);
```

Previous()

Description

Returns the previous material in the model.

Arguments

No arguments

Returns

Material object (or null if there are no more materials in the model).

Return type

Material

Example

To get the material in model m before material m:

```
var m = m.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the materials in the model.

Arguments

- **Model** (Model)

Model that all materials will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the materials in model m, from 1000000:

```
Material.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged materials in the model.

Arguments

- **Model** (Model)

Model that all the flagged materials will be renumbered in

- **flag** (Flag)

Flag set on the materials that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the materials in model m flagged with f, from 1000000:

```
Material.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select materials using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting materials

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only materials from that model can be selected. If the argument is a Flag then only materials that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any materials can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of materials selected or null if menu cancelled

Return type

Number

Example

To select materials from model m, flagging those selected with flag f, giving the prompt 'Select materials':

```
Material.Select(f, 'Select materials', m);
```

To select materials, flagging those selected with flag f but limiting selection to materials flagged with flag l, giving the prompt 'Select materials':

```
Material.Select(f, 'Select materials', l);
```

SetAddDamageGissmoData(data*[object]*)

Description

Sets the *MAT_ADD_DAMAGE_GISSMO data of material.

Arguments

- **data** (object)

Data returned from Material.GetAddDamageGissmoData

Object has the following properties:

Name	Type	Description
biaxf	real	Reduction factor for regularization at triaxiality=2/3
dcrit	real	Damage treshold value
dmgexp	real	Exponent for nonlinear damage accumalation
dtyp	real	Flag for GISSMO damage type
ecrit	real/integer	Critical plastic strain (Curve/ table ID if negative)
fadexp	real/integer	Exponent for damage-related stress fadeout (Curve/ table ID if negative)
hisvn	real	History variable used to evaluate th 3-D table LCSDG
instf	integer	Flag for governing the behavior of instability measure F and fading exponent FADEXP
lcdlim	integer	Curve ID: damage limit values as a function of triaxiality
lcregd	integer	Curve/ table ID (positive) or Table ID (negative): Element-size dependent fading exponent
lcsdg	integer	Curve/ table ID (positive) or Function ID (negative): Failure strain curve/table or function
lcsoft	integer	Soft reduction factor for failure strain in crashfront elements.
lcsrs	integer	Curve/ table ID: Failure strain rate scaling factor v/s strain rate
lp2bi	real	Option to use bending indicator instead of the Lode parameter
midfail	integer	Mid-plane failure option for shell elements and GISSMO
numfip	real	Number of failed integration points prior to element deletion
refsz	real	Reference element size
rgtr1	real	First triaxiality value for optional "tub-shaped" regularization
rgtr2	real	Second triaxiality value for optional "tub-shaped" regularization
shrf	real	Reduction factor for regularization at triaxiality=0

soft	real	Softening reduction factor for failure strain in crashfront elements
stochastic	logical	stochastic = true if _STOCHASTIC is ON. Otherwise, _STOCHASTIC is OFF
volfrac	real	Volume fraction required to fail before element deletion

Returns

No return value

Example

To set the value of MIDFAIL Damage Gissmo for material m to be 3:

```
var data = m.GetAddDamageGissmoData();
data.midfail = 3;
m.SetAddDamageGissmoData(data);
```

SetAddErosionData(data*[object]*)

Description

Sets the *MAT_ADD_EROSION data of material. Note that this method does not support pre-R11 properties.

Arguments

- **data** (object)

Data returned from Material.GetAddErosionData.

Object has the following properties:

Name	Type	Description
dteflt	real	Time period for the low pass filter
dtmin	real	Minimum time step size at failure
effeps	real	Maximum effective strain at failure
engcrt	real	Critical energy for nonlocal failure criterion
epssh	real	Shear strain at failure
epsthin	real	Thinning strain at failure for shells
excl	real	The exclusion number
failtm	real	Failure time
idam	integer	Flag for damage model
impulse	real	Stress impulse for failure
lceps12	integer	Load curve ID defining in-plane shear strain limit vs elem size
lceps13	integer	Load curve ID defining through-thickness shear strain limit vs elem size
lcepsmx	integer	Load curve ID defining in-plane major strain limit vs elem size
lcfld	integer	Curve (negative) or table (positive) ID: Forming limit diagram
lcregd	integer	Curve ID: Element-size dependent fading exponent
mneps	real	Minimum principal strain at failure
mnpres	real	Pressure at failure
mxeps	real/integer	Principal strain at failure (curve ID if negative)
mxpres	real	Maximum pressure at failure
mxtmp	real	Maximum temperature at failure

ncs	real	Number of failure conditions to satisfy before failure occurs
nsff	real	Number of explicit time step cycles for stress fade-out used in the LCFLD criterion
numfip	real	Number of failed integration points prior to element deletion
radcrt	real	Critical radius for nonlocal failure criterion
sigp1	real	Principal stress at failure
sigth	real	Threshold stress
sigvm	real/integer	Equivalent stress at failure (curve ID if negative)
voleps	real	Volumetric strain at failure
volfrac	real	The volume fraction required to fail before the element is deleted

Returns

No return value

Example

To set the value of EXCL Erosion for material m to be 1.25:

```
var data = m.GetAddErosionData();
data.excl = 1.25;
m.SetAddErosionData(data);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the material.

Arguments

- **flag** (Flag)

Flag to set on the material

Returns

No return value

Example

To set flag f for material m:

```
m.SetFlag(f);
```

SetPropertyByIndex(index*[integer]*, value*[integer/real for numeric properties, string for character properties]*)

Description

Sets the value of property at index *index* for this Material object

Arguments

- **index** (integer)

The index of the property value to set. (the number of properties can be found from properties) **Note that indices start at 0**. There is no link between indices and rows/columns so adjacent fields on a line for a material may not have adjacent indices.

- **value** (integer/real for numeric properties, string for character properties)

The value of the property to set.

Returns

No return value

Example

To set the property at index 2, for material m to be 1.234:

```
m.SetPropertyByIndex(2, 1.234);
```

SetPropertyByName(acronym*[string]*, value*[integer/real for numeric properties, string for character properties]*)

Description

Sets the value of property string *acronym* for this Material object

Arguments

- **acronym** (string)

The acronym of the property value to set

- **value** (integer/real for numeric properties, string for character properties)

The value of the property to set.

Returns

No return value

Example

To set the value of RO for material m to be 7.89e-9:

```
m.SetPropertyByName("RO", 7.89e-9);
```

SetPropertyByRowCol(row*[integer]*, col*[integer]*, value*[integer/real for numeric properties, string for character properties]*)

Description

Sets the value of the property for row and col for this Material object.**Note that rows and columns start at 0**.

Arguments

- **row** (integer)

The row of the property value to set

- **col** (integer)

The column of the property value to set

- **value** (integer/real for numeric properties, string for character properties)

The value of the property to set.

Returns

No return value

Example

To set the value of the property at row 0, column 1 for material m to be 7.89e-9:

```
m.SetPropertyByRowCol(0, 1, 7.89e-9);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the material. The material will be sketched until you either call Material.Unsketch(), Material.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the material is sketched. If omitted redraw is true. If you want to sketch several materials and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch material m:

```
m.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged materials in the model. The materials will be sketched until you either call Material.Unsketch(), Material.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged materials will be sketched in

- **flag** (Flag)

Flag set on the materials that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the materials are sketched. If omitted redraw is true. If you want to sketch flagged materials several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all materials flagged with flag in model m:

```
Material.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of materials in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing materials should be counted. If false or omitted referenced but undefined materials will also be included in the total.

Returns

number of materials

Return type

Number

Example

To get the total number of materials in model m:

```
var total = Material.Total(m);
```

Unblank()

Description

Unblanks the material

Arguments

No arguments

Returns

No return value

Example

To unblank material m:

```
m.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the materials in the model.

Arguments

- **Model** (Model)

Model that all materials will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the materials in model m:

```
Material.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged materials in the model.

Arguments

- **Model** (Model)

Model that the flagged materials will be unblanked in

- **flag** (Flag)

Flag set on the materials that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the materials in model m flagged with f:

```
Material.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the materials in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all materials will be unset in

- **flag** (Flag)

Flag to unset on the materials

Returns

No return value

Example

To unset the flag f on all the materials in model m:

```
Material.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the material.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the material is unsketched. If omitted redraw is true. If you want to unsketch several materials and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch material m:

```
m.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all materials.

Arguments

- **Model** (Model)

Model that all materials will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the materials are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all materials in model m:

```
Material.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged materials in the model.

Arguments

- **Model** (Model)

Model that all materials will be unsketched in

- **flag** (Flag)

Flag set on the materials that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the materials are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all materials flagged with flag in model m:

```
Material.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

Material object.

Return type

Material

Example

To check if Material property m.example is a parameter by using the Material.GetParameter() method:

```
if (m.ViewParameters().GetParameter(m.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for material with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The material object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for material m:

```
m.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for material m:

```
m.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for material. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for material m:

```
m.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this material.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for material m:

```
var xrefs = m.Xrefs();
```

YieldStress()

Description

Get Yield stress for the material.

Arguments

No arguments

Returns

real

Return type

Number

Example

To get Yield stress for material m:

```
var yield = m.YieldStress();
```

YoungsModulus()

Description

Get Youngs modulus for the material.

Arguments

No arguments

Returns

real

Return type

Number

Example

To get Youngs modulus for material m:

```
var e = m.YoungsModulus();
```

---

toString()

Description

Creates a string containing the material data in keyword format. Note that this contains the keyword header and the keyword cards. See also Material.Keyword() and Material.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for material m in keyword format

```
var s = m.toString();
```

Node class

The Node class gives you access to node cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Merge(Model*[Model]*, flag*[Flag]*, dist*[real]*, label (optional)*[integer]*, position (optional)*[integer]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- ExtractColour()
- Flagged(flag*[Flag]*)
- GetAttachedShells(recursive (optional)*[boolean]*)
- GetComments()
- GetFreeEdgeNodes()
- GetInitialVelocities()
- GetParameter(prop*[string]*)
- GetReferenceGeometry()
- Keyword()
- KeywordCards()
- Next()
- NodalMass()
- Previous()

- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

Node constants

Name	Description
Node.SCALAR	Node is *NODE_SCALAR.
Node.SCALAR_VALUE	Node is *NODE_SCALAR_VALUE.

Node properties

Name	Type	Description
colour	Colour	The colour of the node
exists (read only)	logical	true if node exists, false if referred to but not defined.
include	integer	The Include file number that the node is in.
label	integer	Node number. Also see the nid property which is an alternative name for this.
model (read only)	integer	The Model number that the node is in.
ndof	integer	Number of degrees of freedom (SCALAR and SCALAR_VALUE only).
nid	integer	Node number. Also see the label property which is an alternative name for this.
rc	integer	Rotational constraint (0-7)
scalar	integer	The type of the node. Can be false (*NODE), Node.SCALAR (*NODE_SCALAR) or Node.SCALAR_VALUE (*NODE_SCALAR_VALUE)
tc	integer	Translational constraint (0-7)
x	real	X coordinate
x1	integer	Initial value of 1st degree of freedom (SCALAR_VALUE only).
x2	integer	Initial value of 2nd degree of freedom (SCALAR_VALUE only).
x3	integer	Initial value of 3rd degree of freedom (SCALAR_VALUE only).
y	real	Y coordinate
z	real	Z coordinate

Detailed Description

The Node class allows you to create, modify, edit and manipulate node cards. See the documentation below for more details.

Constructor

new Node(Model*[Model]*, nid*[integer]*, x*[real]*, y*[real]*, z*[real]*, tc (optional)*[integer]*, rc (optional)*[integer]*)

Description

Create a new Node object.

Arguments

- **Model** (Model)

Model that node will be created in

- **nid** (integer)

Node number

- **x** (real)

X coordinate

- **y** (real)

Y coordinate

- **z** (real)

Z coordinate

- **tc (optional)** (integer)

Translational constraint (0-7). If omitted tc will be set to 0.

- **rc (optional)** (integer)

Rotational constraint (0-7). If omitted rc will be set to 0.

Returns

Node object

Return type

Node

Example

To create a new node in model m with label 100, at coordinates (20, 40, 10)

```
var n = new Node(m, 100, 20, 40, 10);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a node.

Arguments

- **Comment** (Comment)

Comment that will be attached to the node

Returns

No return value

Example

To associate comment c to the node n:

```
n.AssociateComment(c);
```

Blank()

Description

Blanks the node

Arguments

No arguments

Returns

No return value

Example

To blank node n:

```
n.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the nodes in the model.

Arguments

- **Model** (Model)

Model that all nodes will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the nodes in model m:

```
Node.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged nodes in the model.

Arguments

- **Model** (Model)

Model that all the flagged nodes will be blanked in

- **flag** (Flag)

Flag set on the nodes that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the nodes in model m flagged with f:

```
Node.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the node is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if node n is blanked:

```
if (n.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse node n:

```
n.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the node.

Arguments

- **flag** (Flag)

Flag to clear on the node

Returns

No return value

Example

To clear flag f for node n:

```
n.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the node. The target include of the copied node can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

Node object

Return type

Node

Example

To copy node n into node z:

```
var z = n.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a node

Arguments

- **Model** (Model)

Model that the node will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

Node object (or null if not made).

Return type

Node

Example

To start creating a node in model m:

```
var n = Node.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a node.

Arguments

- **Comment** (Comment)

Comment that will be detached from the node

Returns

No return value

Example

To detach comment c from the node n:

```
n.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit node n:

```
n.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for node with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The node object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for node n:

```
n.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for node n:

```
n.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for node. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for node n:

```
n.Error("My custom error");
```

ExtractColour()

Description

Extracts the **actual** colour used for node.
By default in PRIMER many entities such as elements get their colour automatically from the part that they are in. PRIMER cycles through 13 default colours based on the label of the entity. In this case the node colour property will return the value Colour.PART instead of the actual colour. This method will return the actual colour which is used for drawing the node.

Arguments

No arguments

Returns

colour value (integer)

Return type

Number

Example

To return the colour used for drawing node n:

```
var colour = n.ExtractColour();
```

First(Model*[Model]*) [static]

Description

Returns the first node in the model.

Arguments

- **Model** (Model)

Model to get first node in

Returns

Node object (or null if there are no nodes in the model).

Return type

Node

Example

To get the first node in model m:

```
var n = Node.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free node label in the model. Also see Node.LastFreeLabel(), Node.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free node label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

Node label.

Return type

Number

Example

To get the first free node label in model m:

```
var label = Node.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the nodes in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all nodes will be flagged in

- **flag** (Flag)

Flag to set on the nodes

Returns

No return value

Example

To flag all of the nodes with flag f in model m:

```
Node.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the node is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the node

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if node n has flag f set on it:

```
if (n.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each node in the model.
Note that ForEach has been designed to make looping over nodes as fast as possible and so has some limitations. Firstly, a single temporary Node object is created and on each function call it is updated with the current node data. This means that you should not try to store the Node object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new nodes inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all nodes are in

- **func** (function)

Function to call for each node

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the nodes in model m:

```
Node.ForEach(m, test);
function test(n)
{
// n is Node object
}
```

To call function test for all of the nodes in model m with optional object:

```
var data = { x:0, y:0 };
Node.ForEach(m, test, data);
function test(n, extra)
{
// n is Node object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Node objects or properties for all of the nodes in a model in PRIMER. If the optional property argument is not given then an array of Node objects is returned. If the property argument is given, that property value for each node is returned in the array instead of a Node object

Arguments

- **Model** (Model)

Model to get nodes from

- **property (optional)** (string)

Name for property to get for all nodes in the model

Returns

Array of Node objects or properties

Return type

Array

Example

To make an array of Node objects for all of the nodes in model m:

```
var a = Node.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each node in model m:

```
var a = Node.GetAll(m, 'foo');
```

GetAttachedShells(recursive (optional)*[boolean]*)

Description

Returns the shells that are attached to the node.

Arguments

- **recursive (optional)** (boolean)

If recursive is false then only the shells actually attached to the node will be returned (this could also be done by using the Xrefs class but this method is provided for convenience. If recursive is true then PRIMER will keep finding attached shells until no more can be found. If omitted recursive will be false.

Returns

Array of Shell objects (or null if there are no attached shells).

Return type

Array

Example

To find the shells attached to node n, growing the selection until no more shells can be found:

```
var shell_array = n.GetAttachedShells(true);
```

GetComments()

Description

Extracts the comments associated to a node.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the node n:

```
var comm_array = n.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of Node objects for all of the flagged nodes in a model in PRIMER If the optional property argument is not given then an array of Node objects is returned. If the property argument is given, then that property value for each node is returned in the array instead of a Node object

Arguments

- **Model** (Model)

Model to get nodes from

- **flag** (Flag)

Flag set on the nodes that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged nodes in the model

Returns

Array of Node objects or properties

Return type

Array

Example

To make an array of Node objects for all of the nodes in model m flagged with f:

```
var n = Node.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the nodes in model m flagged with f:

```
var a = Node.GetFlagged(m, f, 'foo');
```

GetFreeEdgeNodes()

Description

If the node is on a shell free edge and that edge forms a loop like the boundary of a hole, then GetFreeEdgeNodes returns all of the nodes on the hole/boundary in order.

Note that a free edge is a shell edge which is only used by one shell, whereas edges in the middle of a shell part will have got more than one adjacent shell and are therefore not free edges. If every node on a boundary belongs to exactly two free edges, then this function returns the array as described. In more involved combinatorics of shells, for example multiple parts sharing nodes along their boundaries, there can be one, three or more free edges at a node, and this function should not be used.

If you only need to know whether or not a node is on a free edge, you should find the shells attached to it by cross references with Xrefs.GetItemID and see whether these shells have got other nodes in common as well. If nodes along an edge of a shell only appear in that one shell, this is a free edge.

Arguments

No arguments

Returns

Array of Node objects (or null if not on a shell free edge).

Return type

Array

Example

To find all the nodes on the hole/boundary that node n is on:

```
var node_array = n.GetFreeEdgeNodes();
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the Node object for a node ID.

Arguments

- **Model** (Model)

Model to find the node in

- **number** (integer)

number of the node you want the Node object for

Returns

Node object (or null if node does not exist).

Return type

Node

Example

To get the Node object for node 100 in model m

```
var n = Node.GetFromID(m, 100);
```

GetInitialVelocities()

Description

Returns the initial velocity of the node. You need to be sure the field nvels of the node is populate before to use GetInitialVelocities. To do so you can use Model.PopNodeVels .

Arguments

No arguments

Returns

Array containing the 3 translational and 3 rotational velocity values.

Return type

Array

Example

To get the initial velocity of the node n:

```
var vel = n.GetInitialVelocities();
```

GetParameter(prop*[string]*)

Description

Checks if a Node property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the Node.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

node property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if Node property n.example is a parameter:

```
Options.property_parameter_names = true;
if (n.GetParameter(n.example) ) do_something...
Options.property_parameter_names = false;
```

To check if Node property n.example is a parameter by using the GetParameter method:

```
if (n.ViewParameters().GetParameter(n.example) ) do_something...
```

GetReferenceGeometry()

Description

Returns the airbag reference geometry of the node

Arguments

No arguments

Returns

The reference geometry ID of the node (or 0 if it hasn't got any)

Return type

Number

Example

To get the reference geometry of the node n:

```
var a = n.GetReferenceGeometry();
```

Keyword()

Description

Returns the keyword for this node (*NODE, *NODE_SCALAR or *NODE_SCALAR_VALUE). **Note that a carriage return is not added**. See also Node.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for node n:

```
var key = n.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the node. **Note that a carriage return is not added**. See also Node.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for node n:

```
var cards = n.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last node in the model.

Arguments

- **Model** (Model)

Model to get last node in

Returns

Node object (or null if there are no nodes in the model).

Return type

Node

Example

To get the last node in model m:

```
var n = Node.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free node label in the model. Also see Node.FirstFreeLabel(), Node.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free node label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

Node label.

Return type

Number

Example

To get the last free node label in model m:

```
var label = Node.LastFreeLabel(m);
```

Merge(Model*[Model]*, flag*[Flag]*, dist*[real]*, label (optional)*[integer]*, position (optional)*[integer]*) [static]

Description

Attempts to merge nodes flagged with flag for a model in PRIMER. Merging nodes on *AIRBAG_SHELL_ REFERENCE_GEOMETRY can be controlled by using Options.node_replace_asrg. Also see Model.MergeNodes().

Arguments

- **Model** (Model)

Model that the nodes will be merged in

- **flag** (Flag)

Flag set on nodes to nodes

- **dist** (real)

Nodes closer than dist will be potentially merged.

- **label (optional)** (integer)

Label to keep after merge. If > 0 then highest label kept. If <= 0 then lowest kept. If omitted the lowest label will be

kept.

- **position (optional)** (integer)

Position to merge at. If > 0 then merged at highest label position. If < 0 then merged at lowest label position. If 0 then merged at midpoint. If omitted the merge will be done at the lowest label.

Returns

The number of nodes merged

Return type

Number

Example

To (try to) merge nodes in model m flagged with flag f, with a distance of 0.1:

```
Node.Merge(m, f, 0.1);
```

Next()

Description

Returns the next node in the model.

Arguments

No arguments

Returns

Node object (or null if there are no more nodes in the model).

Return type

Node

Example

To get the node in model m after node n:

```
var n = n.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) node label in the model. Also see Node.FirstFreeLabel(), Node.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free node label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

Node label.

Return type

Number

Example

To get the next free node label in model m:

```
var label = Node.NextFreeLabel(m);
```

NodalMass()

Description

Get the mass of a node. This will be the sum of the structural element mass attached to the node plus any lumped mass. If called on the node of a PART_INERTIA or NRBC_INERTIA, this function will return the mass of the part/nrbc, as 'nodal mass' has no meaning in this context.

Arguments

No arguments

Returns

real

Return type

Number

Example

To get the mass for node n:

```
var mass = n.NodalMass();
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a node.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only nodes from that model can be picked. If the argument is a Flag then only nodes that are flagged with *limit* can be selected. If omitted, or null, any nodes from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

Node object (or null if not picked)

Return type

Node

Example

To pick a node from model m giving the prompt 'Pick node from screen':

```
var n = Node.Pick('Pick node from screen', m);
```

Previous()

Description

Returns the previous node in the model.

Arguments

No arguments

Returns

Node object (or null if there are no more nodes in the model).

Return type

Node

Example

To get the node in model m before node n:

```
var n = n.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the nodes in the model.

Arguments

- **Model** (Model)

Model that all nodes will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the nodes in model m, from 1000000:

```
Node.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged nodes in the model.

Arguments

- **Model** (Model)

Model that all the flagged nodes will be renumbered in

- **flag** (Flag)

Flag set on the nodes that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the nodes in model m flagged with f, from 1000000:

```
Node.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select nodes using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting nodes

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only nodes from that model can be selected. If the argument is a Flag then only nodes that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any nodes can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of nodes selected or null if menu cancelled

Return type

Number

Example

To select nodes from model m, flagging those selected with flag f, giving the prompt 'Select nodes':

```
Node.Select(f, 'Select nodes', m);
```

To select nodes, flagging those selected with flag f but limiting selection to nodes flagged with flag l, giving the prompt 'Select nodes':

```
Node.Select(f, 'Select nodes', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the node.

Arguments

- **flag** (Flag)

Flag to set on the node

Returns

No return value

Example

To set flag f for node n:

```
n.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the node. The node will be sketched until you either call Node.Unsketch(), Node.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the node is sketched. If omitted redraw is true. If you want to sketch several nodes and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch node n:

```
n.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged nodes in the model. The nodes will be sketched until you either call Node.Unsketch(), Node.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged nodes will be sketched in

- **flag** (Flag)

Flag set on the nodes that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the nodes are sketched. If omitted redraw is true. If you want to sketch flagged nodes several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all nodes flagged with flag in model m:

```
Node.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of nodes in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing nodes should be counted. If false or omitted referenced but undefined nodes will also be included in the total.

Returns

number of nodes

Return type

Number

Example

To get the total number of nodes in model m:

```
var total = Node.Total(m);
```

Unblank()

Description

Unblanks the node

Arguments

No arguments

Returns

No return value

Example

To unblank node n:

```
n.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the nodes in the model.

Arguments

- **Model** (Model)

Model that all nodes will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the nodes in model m:

```
Node.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged nodes in the model.

Arguments

- **Model** (Model)

Model that the flagged nodes will be unblanked in

- **flag** (Flag)

Flag set on the nodes that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the nodes in model m flagged with f:

```
Node.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the nodes in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all nodes will be unset in

- **flag** (Flag)

Flag to unset on the nodes

Returns

No return value

Example

To unset the flag f on all the nodes in model m:

```
Node.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the node.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the node is unsketched. If omitted redraw is true. If you want to unsketch several nodes and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch node n:

```
n.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all nodes.

Arguments

- **Model** (Model)

Model that all nodes will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the nodes are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all nodes in model m:

```
Node.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged nodes in the model.

Arguments

- **Model** (Model)

Model that all nodes will be unsketched in

- **flag** (Flag)

Flag set on the nodes that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the nodes are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all nodes flagged with flag in model m:

```
Node.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

Node object.

Return type

Node

Example

To check if Node property n.example is a parameter by using the Node.GetParameter() method:

```
if (n.ViewParameters().GetParameter(n.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for node with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The node object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for node n:

```
n.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for node n:

```
n.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for node. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for node n:

```
n.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this node.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for node n:

```
var xrefs = n.Xrefs();
```

---

toString()

Description

Creates a string containing the node data in keyword format. Note that this contains the keyword header and the keyword cards. See also Node.Keyword() and Node.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for node n in keyword format

```
var s = n.toString();
```

Parameter class

The Parameter class allows you to access the parameters in a model. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- AutoReorder(Model*[Model]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- GetAll(Model*[Model]*)
- GetAllOfName(Model*[Model]*)
- GetFromName(Model*[Model]*, parameter name*[string]*)
- SaveAll(Model*[Model]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UpdateAll(Model*[Model]*)

Member functions

- ClearFlag(flag*[Flag]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Evaluate()
- Flagged(flag*[Flag]*)
- Keyword()
- KeywordCards()
- SetFlag(flag*[Flag]*)
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

Parameter constants

Name	Description
Parameter.CHARACTER	Parameter is a character.
Parameter.INTEGER	Parameter is an integer.
Parameter.LOCAL	Parameter has _LOCAL suffix (used in suffix argument for constructor).
Parameter.MUTABLE	Parameter has _MUTABLE suffix (used in suffix argument for constructor).
Parameter.NOECHO	Parameter has _NOECHO suffix (used in suffix argument for constructor).
Parameter.REAL	Parameter is a real.

Parameter properties

Name	Type	Description
expression (read only)	logical	true if this parameter is a *PARAMETER_EXPRESSION, false otherwise.
include	integer	The Include file number that the parameter is in.

local	logical	true if this parameter is a *PARAMETER_... _LOCAL, false otherwise.
model	integer	The Model number that the parameter is in.
mutable	logical	true if this parameter is a *PARAMETER_... _MUTABLE, false otherwise.
name (read only)	string	Parameter name.
noecho	logical	true if this parameter is a *PARAMETER_... _NOECHO, false otherwise.
type (read only)	constant	Can be Parameter.INTEGER, Parameter.REAL or Parameter.CHARACTER.
value	integer/real/string	Parameter value. The value will be a string for parameter expressions, or a number for normal parameters. By default when a parameter value is changed PRIMER will re-evaluate and update all of the parameters in the model as changing this parameter could cause others to change because of parameter expressions. There could be some situations where changing parameters one at a time could cause problems with re-evaluation. For example, changing parameter A could temporarily cause a division by zero when re-evaluating parameter expression B until parameter C is changed. In this case the automatic re-evaluation can be prevented by using Parameter.SaveAll and Parameter.UpdateAll.

Detailed Description

The Parameter class allows to create and query parameters in a model. See the documentation below for more details.

Constructor

new Parameter(Model*[Model]*, name*[string]*, type*[constant]*, expression*[boolean]*, value*[integer/real/string]*, suffix (optional)*[constant]*)

Description

Create a new Parameter object.

Arguments

- **Model** (Model)

Model that parameter will be created in

- **name** (string)

Parameter name

- **type** (constant)

Can be Parameter.INTEGER, Parameter.REAL or Parameter.CHARACTER.

- **expression** (boolean)

true if *PARAMETER_EXPRESSION, false otherwise.

- **value** (integer/real/string)

Parameter value. The value will be a string for character parameters or parameter expressions, or a number for integer or real parameters.

- **suffix (optional)** (constant)

Keyword suffix Can be Parameter.LOCAL for *PARAMETER_..._LOCAL, Parameter.MUTABLE for *PARAMETER_..._MUTABLE, or Parameter.NOECHO for *PARAMETER_..._NOECHO. These may be bitwise ORed together, ie Parameter.LOCAL | Parameter.MUTABLE | Parameter.NOECHO. If omitted the parameter will not be local or mutable.

Returns

Parameter object

Return type

Parameter

Example

To create a new real parameter THK in model m with value 5.0

```
var p = new Parameter(m, "THK", Parameter.REAL, false, 5.0);
```

To create a new LOCAL integer parameter INDEX in model m with value 3

```
var p = new Parameter(m, "INDEX", Parameter.INTEGER, false, 3, Parameter.LOCAL);
```

Details of functions

AutoReorder(Model[*Model*]) [static]

Description

Auto Reorders all the parameters in the model.

Arguments

- **Model** (Model)

Model that contains all parameters that will be re-ordered

Returns

No return value

Example

To auto-reorder all parameters in model m:

```
Parameter.AutoReorder(m);
```

ClearFlag(flag[*Flag*])

Description

Clears a flag on the parameter.

Arguments

- **flag** (Flag)

Flag to clear on the parameter

Returns

No return value

Example

To clear flag f for parameter p:

```
p.ClearFlag(f);
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for a parameter with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The parameter object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for parameter p:

```
p.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for parameter p:

```
p.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for a parameter. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for parameter p:

```
p.Error("My custom error");
```

Evaluate()

Description

Evaluates a parameter expression, updating the evaluated value stored in PRIMER and returns the value. If the parameter is not an expression then the parameter value will just be returned.
If evaluating the expression cannot be done because of an error (e.g. dividing by zero) an exception will be thrown.

Arguments

No arguments

Returns

number (real and integer parameters) or string (character parameters)

Return type

Number

Example

To evaluate parameter p:

```
var value = p.Evaluate();
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the parameters in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all parameters will be flagged in

- **flag** (Flag)

Flag to set on the parameters

Returns

No return value

Example

To flag all of the parameters with flag f in model m:

```
Parameter.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the parameter is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the parameter

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if parameter p has flag f set on it:

```
if (p.Flagged(f) ) do_something...
```

GetAll(Model[*Model*]) [static]

Description

Returns an array of Parameter objects for all of the parameters in a model in Primer

Arguments

- **Model** (Model)

Model to get parameters from

Returns

Array of Parameter objects

Return type

Array

Example

To make an array of Parameter objects for all of the parameters in model m

```
var p = Parameter.GetAll(m);
```

GetAllOfName(Model[*Model*]) [static]

Description

Returns an array of Parameter objects for all parameters in a model matching Name. If none are found that match it will return NULL. (Multiple parameters of the same name may exist if they use the _LOCAL or _MUTABLE suffices. PRIMER will also store multiple illegal instances of parameter name, using the instance as determined by the PARAMETER_DUPLICATION card.)

Arguments

- **Model** (Model)

Model to get parameters from

Returns

Array of Parameter objects

Return type

Array

Example

To make an array of Parameter objects for all of the parameters of name in model m

```
var p = Parameter.GetAllOfName(m, name);
```

GetFromName(Model*[Model]*, parameter name*[string]*) [static]

Description

Returns the stored Parameter object for a parameter name. WARNING: if more than one parameter Name exists (eg _LOCAL, _MUTABLE) then only the first occurrence is returned. To return all parameters matching Name use GetAllOfName() instead.

Arguments

- **Model** (Model)

Model to find the parameter in

- **parameter name** (string)

name of the parameter you want the Parameter object for

Returns

Parameter object (or null if parameter does not exist).

Return type

Parameter

Example

To get the Parameter object for parameter "THK" in model m

```
var p = Parameter.GetFromName(m, "THK");
```

Keyword()

Description

Returns the keyword for this parameter (*PARAMETER, *PARAMETER_EXPRESSION). **Note that a carriage return is not added**. See also Parameter.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for parameter p:

```
var key = p.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the parameter. **Note that a carriage return is not added**. See also Parameter.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for parameter p:

```
var cards = p.KeywordCards();
```

SaveAll(Model[*Model*]) [static]

Description

Saves the current status and values of all of the parameters in the model. Calling this will also have the effect of turning off re-evaluating and updating of all parameters in the model when a parameter value is changed.
To update several parameters in a model without re-evaluating all the parameters after each one is changed first call this, then update all of the parameter values, and then call Parameter.UpdateAll to apply the update.
Parameter.SaveAll **must** be called before using Parameter.UpdateAll.

Arguments

- **Model** (Model)

Model that the parameters will be saved in

Returns

No return value

Example

To save the status of all of the parameters in model m:

```
Parameter.SaveAll(m);
```

SetFlag(flag[*Flag*])

Description

Sets a flag on the parameter.

Arguments

- **flag** (Flag)

Flag to set on the parameter

Returns

No return value

Example

To set flag f for parameter p:

```
p.SetFlag(f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the parameters in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all parameters will be unset in

- **flag** (Flag)

Flag to unset on the parameters

Returns

No return value

Example

To unset the flag f on all of the parameters in model m:

```
Parameter.UnflagAll(m, f);
```

UpdateAll(Model*[Model]*) [static]

Description

Updates all of the parameters in the model after saving the state of all parameters using Parameter.SaveAll and modifying the parameter values. As parameter re-evaluation has been suppressed by Parameter.SaveAll you should ensure that all parameters in the model can be evaluated correctly before calling this to ensure that there are no errors. If any of the parameters cannot be evaluated then the values saved in Parameter.SaveAll will be restored, the update will be aborted and an exception thrown. Calling this will also have the effect of turning back on re-evaluating and updating of all parameters in the model when a parameter value is changed.
Parameter.SaveAll **must** be called before this method can be used.

Arguments

- **Model** (Model)

Model that the parameters will be updated in

Returns

No return value

Example

To update all of the parameters in model m:

```
Parameter.UpdateAll(m);
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for a parameter with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The parameter object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for parameter p:

```
p.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for parameter p:

```
p.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for a parameter. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for parameter p:

```
p.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this parameter.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for parameter p:

```
var xrefs = p.Xrefs();
```

---

toString()

Description

Creates a string containing the parameter data in keyword format. Note that this contains the keyword header and the keyword cards. See also Parameter.Keyword() and Parameter.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for parameter p in keyword format

```
var s = p.toString();
```

Part class

The Part class gives you access to part cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- AllTableProperties(Model*[Model]*)
- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- FlagVisible(Model*[Model]*, flag*[Flag]*)
- FlaggedTableProperties(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- MeasurePartToPart(part1*[Part]*, part2*[Part]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- CentreOfGravity(options (optional)*[object]*)
- ClearFlag(flag*[Flag]*)
- ClosestNode(x*[real]*, y*[real]*, z*[real]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- ExtractColour()
- Flagged(flag*[Flag]*)
- GetComments()
- GetCompositeData(ipt*[integer]*)
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Mass()

- MaxMin()
- Next()
- Previous()
- RemoveCompositeData(ipt*[integer]*)
- SetCompositeData(ipt*[integer]*, mid*[integer]*, thick*[real]*, beta*[real]*, tmid (optinal)*[integer]*, plyid (optional)*[integer]*, shrfac (optional)*[real]*)
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- TableProperties()
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

Part properties

Name	Type	Description
adpopt	integer	Adaptivity flag
ansid	integer	Attachment node set ID
attachment_nodes	logical	If _ATTACHMENT_NODES option is set. Can be true or false
averaged	logical	If _AVERAGED option is set. Can be true or false
cadname	string	CAD name stored for Part (or null if doesn't exist). This property is only used by PRIMER.
cid	integer	Coordinate system number
cmsn	integer	CAL3D/MADYMO number
colour	Colour	The colour of the part
composite	logical	If _COMPOSITE option is set. Can be true or false
composite_long	logical	If _COMPOSITE_LONG option is set. Can be true or false
contact	logical	If _CONTACT option is set. Can be true or false
dc	real	Exponential decay coefficient
element_type (read only)	string	The type of elements the Part contains. e.g. "SHELL", "SOLID" or null if empty/no section.
elform	integer	Element formulation
eosid	integer or string	Equation of state number or character label
exists (read only)	logical	true if part exists, false if referred to but not defined.
fd	real	Dynamic coefficient of friction
fs	real	Static coefficient of friction
grav	integer	Gravity loading
heading	string	Part heading
hgid	integer or string	Hourglass number or character label
hmname	string	Hypermesh comment read from keyword file for Part (or null if doesn't exist).
iga_shell	logical	If _COMPOSITE_IGA_SHELL option is set. Can be true or false
include	integer	The Include file number that the part is in.

inertia	logical	If _INERTIA option is set. Can be true or false
ircs	integer	Flag for inertia tensor reference coordinate system
irl	integer	Lamina integration rule
ixx	real	Ixx component of inertia tensor
ixy	real	Ixy component of inertia tensor
ixz	real	Ixz component of inertia tensor
iyy	real	Iyy component of inertia tensor
iyz	real	Iyz component of inertia tensor
izz	real	Izz component of inertia tensor
label	integer or string	Part number or character label. Also see the pid property which is an alternative name for this.
marea	real	Non structural mass per unit area
mdep	integer	MADYMO ellipse/plane number
mid	integer or string	Material number or character label
model (read only)	integer	The Model number that the part is in.
movopt	integer	Flag to deactivate moving for merged rigid bodies
nip	integer	Number of integration points (layers) present for _COMPOSITE parts
nloc	integer	Location of reference surface
nodeid	integer	Node ID for centre of rigid body
optt	real	Contact thickness
pid	integer or string	Part number or character label. Also see the label property which is an alternative name for this.
prbf	integer	Print flag for RBDOUT and MATSUM files
print	logical	If _PRINT option is set. Can be true or false
reposition	logical	If _REPOSITION option is set. Can be true or false
rigid (read only)	logical	true if part is rigid, false if deformable.
secid	integer or string	Section number or character label
sft	real	Thickness scale factor
shrf	real	Shear correction factor
ssf	real	Scale factor on default surfA penalty stiffness
thshel	integer	Thermal shell formulation
tm	real	total mass
tmid	integer or string	Thermal material number or character label
transparency	integer	The transparency of the part (0-100) 0% is opaque, 100% is transparent.
tshear	integer	Flag for transverse shear strain distribution
tshell	logical	If _COMPOSITE_TSHELL option is set. Can be true or false
vc	real	Coefficient for viscous friction
vrx	real	x rotational velocity

vry	real	y rotational velocity
vrz	real	z rotational velocity
vtx	real	x translational velocity
vty	real	y translational velocity
vtz	real	z translational velocity
xc	real	x coordinate of centre of mass
xl	real	x coordinate of local x axis
xlip	real	x coordinate of vector in local xy plane
yc	real	y coordinate of centre of mass
yl	real	y coordinate of local x axis
ylip	real	y coordinate of vector in local xy plane
zc	real	z coordinate of centre of mass
zl	real	z coordinate of local x axis
zlip	real	z coordinate of vector in local xy plane

Detailed Description

The Part class allows you to create, modify, edit and manipulate part cards. See the documentation below for more details.

Constructor

new Part(Model*[Model]*, pid*[integer or string]*, secid*[integer or string]*, mid*[integer or string]*, heading (optional)*[string]*)

Description

Create a new Part object.

Arguments

- **Model** (Model)

Model that part will be created in

- **pid** (integer or string)

Part number or character label

- **secid** (integer or string)

Section number or character label

- **mid** (integer or string)

Material number or character label

- **heading (optional)** (string)

Title for the part

Returns

Part object

Return type

Part

Example

To create a new part called 'Example' in model m with label 100, section 1, material 10:

```
var p = new Part(m, 100, 1, 10, 'Example');
```

Details of functions

AllTableProperties(Model[*Model*]) [static]

Description

Returns all of the properties available in the part table for the parts. The table values are returned in an array of objects (an object for each part). The object property names are the same as the table headers but spaces are replaced with underscore characters and characters other than 0-9, a-z and A-Z are removed to ensure that the property name is valid in JavaScript. If a table value is undefined the property value will be the JavaScript undefined value. If the table value is a valid number it will be a number, otherwise the value will returned as a string.

Arguments

- **Model** (Model)

Model that the flagged parts are in

Returns

Array of objects

Return type

Array

Example

To get all of the properties for parts in model m:

```
var properties = Part.AllTableProperties(m);
for (var p=0; p<properties.length; p++)
{
    for (var x in properties[p])
    {
        Message(x+"="+properties[p][x]);
    }
}
```

AssociateComment(Comment[*Comment*])

Description

Associates a comment with a part.

Arguments

- **Comment** (Comment)

Comment that will be attached to the part

Returns

No return value

Example

To associate comment c to the part p:

```
p.AssociateComment(c);
```

Blank()

Description

Blanks the part

Arguments

No arguments

Returns

No return value

Example

To blank part p:

```
p.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the parts in the model.

Arguments

- **Model** (Model)

Model that all parts will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the parts in model m:

```
Part.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged parts in the model.

Arguments

- **Model** (Model)

Model that all the flagged parts will be blanked in

- **flag** (Flag)

Flag set on the parts that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the parts in model m flagged with f:

```
Part.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the part is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if part p is blanked:

```
if (p.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse part p:

```
p.Browse();
```

CentreOfGravity(options (optional)*[object]*)

Description

Returns the centre of gravity for a part.

Arguments

- **options (optional)** (object)

Options specifying how the mass calculation should be done.

Object has the following properties:

Name	Type	Description
constrainedparts (optional)	boolean	Mass of rigid lead part includes mass of its constrained parts. On by default.
lumpedmass (optional)	boolean	Lumped mass is included for deformable parts. Off by default.
nrbmass (optional)	boolean	NRB mass is included for deformable parts. Off by default. (tranfermass:true required for this option)
plot (optional)	boolean	Plot CofG.
skipconstrained (optional)	boolean	Constrained rigid part is assigned zero mass (if constrainedparts = true). On by default.
timestepmass (optional)	boolean	Timestep added mass is included for deformable parts. Off by default.
transfermass (optional)	boolean	Mass of deformable nodes attached to rigid part/nrb is transferred. On by default.

Returns

An array containing the x, y and z coordinates for the CofG.

Return type

Array

Example

To get the centre of gravity for part p with options configured:

```
var cofg = p.CentreOfGravity({constrainedparts:false, transfermass:true,
lumpedmass:false, nrbmass:true, timestepmass:false, plot:true});
var x = cofg[0];
var y = cofg[1];
var z = cofg[2];
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the part.

Arguments

- **flag** (Flag)

Flag to clear on the part

Returns

No return value

Example

To clear flag f for part p:

```
p.ClearFlag(f);
```

ClosestNode(x*[real]*, y*[real]*, z*[real]*)

Description

Finds the Node on the part closest to a coordinate.

Arguments

- **x** (real)

X coordinate of point

- **y** (real)

Y coordinate of point

- **z** (real)

Z coordinate of point

Returns

ID of Node or null if part has no nodes

Return type

Number

Example

To find the node on part p closest to point (1, 2, 3):

```
var n = p.ClosestNode(1, 2, 3);
```

Copy(range (optional)*[boolean]*)

Description

Copies the part. The target include of the copied part can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

Part object

Return type

Part

Example

To copy part p into part z:

```
var z = p.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a part

Arguments

- **Model** (Model)

Model that the part will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

Part object (or null if not made).

Return type

Part

Example

To start creating a part in model m:

```
var p = Part.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a part.

Arguments

- **Comment** (Comment)

Comment that will be detached from the part

Returns

No return value

Example

To detach comment c from the part p:

```
p.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit part p:

```
p.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for part with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The part object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for part p:

```
p.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for part p:

```
p.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for part. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for part p:

```
p.Error("My custom error");
```

ExtractColour()

Description

Extracts the **actual** colour used for part.
By default in PRIMER many entities such as elements get their colour automatically from the part that they are in. PRIMER cycles through 13 default colours based on the label of the entity. In this case the part colour property will return the value Colour.PART instead of the actual colour. This method will return the actual colour which is used for drawing the part.

Arguments

No arguments

Returns

colour value (integer)

Return type

Number

Example

To return the colour used for drawing part p:

```
var colour = p.ExtractColour();
```

First(Model*[Model]*) [static]

Description

Returns the first part in the model.

Arguments

- **Model** (Model)

Model to get first part in

Returns

Part object (or null if there are no parts in the model).

Return type

Part

Example

To get the first part in model m:

```
var p = Part.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free part label in the model. Also see Part.LastFreeLabel(), Part.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free part label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

Part label.

Return type

Number

Example

To get the first free part label in model m:

```
var label = Part.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the parts in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all parts will be flagged in

- **flag** (Flag)

Flag to set on the parts

Returns

No return value

Example

To flag all of the parts with flag f in model m:

```
Part.FlagAll(m, f);
```

FlagVisible(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all the unblanked parts in the model.

Arguments

- **Model** (Model)

Model for which all unblanked parts will be flagged in

- **flag** (Flag)

Flag to set on the unblanked parts

Returns

No return value

Example

To flag all unblanked parts in model m with flag f:

```
Part.FlagVisible(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the part is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the part

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if part p has flag f set on it:

```
if (p.Flagged(f) ) do_something...
```

FlaggedTableProperties(Model*[Model]*, flag*[Flag]*) [static]

Description

Returns all of the properties available in the part table for the flagged parts. The table values are returned in an array of objects (an object for each part). The object property names are the same as the table headers but spaces are replaced with underscore characters and characters other than 0-9, a-z and A-Z are removed to ensure that the property name is valid in JavaScript. If a table value is undefined the property value will be the JavaScript undefined value. If the table value is a valid number it will be a number, otherwise the value will returned as a string.

Arguments

- **Model** (Model)

Model that the flagged parts are in

- **flag** (Flag)

Flag set on the parts that you want properties for

Returns

Array of objects

Return type

Array

Example

To get all of the properties for parts in model m flagged with f:

```
var properties = Part.FlaggedTableProperties(m, f);
for (var p=0; p<properties.length; p++)
{
    for (var x in properties[p])
    {
        Message(x+"="+properties[p][x]);
    }
}
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each part in the model.
Note that ForEach has been designed to make looping over parts as fast as possible and so has some limitations. Firstly, a single temporary Part object is created and on each function call it is updated with the current part data. This means that you should not try to store the Part object for later use (e.g. in an array) as it is temporary. Secondly, you cannot create new parts inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all parts are in

- **func** (function)

Function to call for each part

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the parts in model m:

```
Part.ForEach(m, test);
function test(p)
{
// p is Part object
}
```

To call function test for all of the parts in model m with optional object:

```
var data = { x:0, y:0 };
Part.ForEach(m, test, data);
function test(p, extra)
{
// p is Part object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Part objects or properties for all of the parts in a model in PRIMER. If the optional property argument is not given then an array of Part objects is returned. If the property argument is given, that property value for each part is returned in the array instead of a Part object

Arguments

- **Model** (Model)

Model to get parts from

- **property (optional)** (string)

Name for property to get for all parts in the model

Returns

Array of Part objects or properties

Return type

Array

Example

To make an array of Part objects for all of the parts in model m:

```
var a = Part.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each part in model m:

```
var a = Part.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a part.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the part p:

```
var comm_array = p.GetComments();
```

GetCompositeData(ipt*[integer]*)

Description

Returns the composite data for an integration point in *PART_COMPOSITE.

Arguments

- **ipt** (integer)

The integration point you want the data for. **Note that integration points start at 0, not 1**.

Returns

An array containing the material id, thickness, beta angle and thermal material values. If the _COMPOSITE_LONG option is set, then the array returned will also contain the ply ID.

Return type

Array

Example

To get the composite data for the 3rd integration point for part p:

```
if (p.composite && p.nip >= 3)
{
    var ipt_data = p.GetCompositeData(2);
}
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of Part objects for all of the flagged parts in a model in PRIMER If the optional property argument is not given then an array of Part objects is returned. If the property argument is given, then that property value for each part is returned in the array instead of a Part object

Arguments

- **Model** (Model)

Model to get parts from

- **flag** (Flag)

Flag set on the parts that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged parts in the model

Returns

Array of Part objects or properties

Return type

Array

Example

To make an array of Part objects for all of the parts in model m flagged with f:

```
var p = Part.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the parts in model m flagged with f:

```
var a = Part.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the Part object for a part ID.

Arguments

- **Model** (Model)

Model to find the part in

- **number** (integer)

number of the part you want the Part object for

Returns

Part object (or null if part does not exist).

Return type

Part

Example

To get the Part object for part 100 in model m

```
var p = Part.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a Part property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the Part.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

part property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if Part property p.example is a parameter:

```
Options.property_parameter_names = true;
if (p.GetParameter(p.example) ) do_something...
Options.property_parameter_names = false;
```

To check if Part property p.example is a parameter by using the GetParameter method:

```
if (p.ViewParameters().GetParameter(p.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this part (*PART, *PART_SCALAR or *PART_SCALAR_VALUE). **Note that a carriage return is not added**. See also Part.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for part p:

```
var key = p.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the part. **Note that a carriage return is not added**. See also Part.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for part p:

```
var cards = p.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last part in the model.

Arguments

- **Model** (Model)

Model to get last part in

Returns

Part object (or null if there are no parts in the model).

Return type

Part

Example

To get the last part in model m:

```
var p = Part.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free part label in the model. Also see Part.FirstFreeLabel(), Part.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free part label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

Part label.

Return type

Number

Example

To get the last free part label in model m:

```
var label = Part.LastFreeLabel(m);
```

Mass()

Description

Returns the mass properties for a part.

Arguments

No arguments

Returns

Object with the following properties:

Name	Type	Description
Assign_Mass	real	Assign mass
Component_Mass	real	Component mass
Dyna_Added_Mass	real	Dyna added mass
Dyna_Part_Mass	real	Dyna part mass
Lumped_Mass	real	Lumped mass
NRB_Mass	real	NRB mass
NS_Mass	real	Non-structural mass
Struct_Mass	real	Structural mass
Transferrd_Mass	real	Transferred mass when deformable meshed to rigid

Return type

object

Example

To get the structural mass for part p:

```
var mprops = p.Mass();
                        var struct_mass = mprops.Struct_Mass;
```

MaxMin()

Description

Returns the max and min bounds of a part

Arguments

No arguments

Returns

An array of numbers containing the xMin, xMax, yMin, yMax, zMin and zMax coordinates for a box bounding the part, or null if the bounds cannot be calculated (e.g. the part has no structural elements)

Return type

Number

Example

To get the bounds for part p:

```
var bounds = p.MaxMin();
if (bounds) {
    xMin = bounds[0];
    xMax = bounds[1];
    yMin = bounds[2];
    yMax = bounds[3];
    zMin = bounds[4];
    zMax = bounds[5];
}
```

MeasurePartToPart(part1*[Part]*, part2*[Part]*) [static]

Description

This static method measures the distance between two part objects contained in the same model or in two different models

Arguments

- **part1** (Part)

Part to measure from

- **part2** (Part)

Part to measure to

Returns

Object with the following properties:

Name	Type	Description
distance	real	Distance between the two parts
vector	Array of reals	Components of distance vector

Return type

object

Example

To measure the distance between part object p1 and part object p2:

```
var m = Part.MeasurePartToPart(p1, p2);
var d = m.distance;
var XComp = m.vector[0];
var YComp = m.vector[1];
var ZComp = m.vector[2];
```

Next()

Description

Returns the next part in the model.

Arguments

No arguments

Returns

Part object (or null if there are no more parts in the model).

Return type

Part

Example

To get the part in model m after part p:

```
var p = p.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) part label in the model. Also see Part.FirstFreeLabel(), Part.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free part label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

Part label.

Return type

Number

Example

To get the next free part label in model m:

```
var label = Part.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a part.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only parts from that model can be picked. If the argument is a Flag then only parts that are flagged with *limit* can be selected. If omitted, or null, any parts from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

Part object (or null if not picked)

Return type

Part

Example

To pick a part from model m giving the prompt 'Pick part from screen':

```
var p = Part.Pick('Pick part from screen', m);
```

Previous()

Description

Returns the previous part in the model.

Arguments

No arguments

Returns

Part object (or null if there are no more parts in the model).

Return type

Part

Example

To get the part in model m before part p:

```
var p = p.Previous();
```

RemoveCompositeData(ipt*[integer]*)

Description

Removes the composite data for an integration point in *PART_COMPOSITE.

Arguments

- **ipt** (integer)

The integration point you want to remove. **Note that integration points start at 0, not 1**.

Returns

No return value.

Example

To remove the composite data for the 3rd integration point for part p:

```
p.RemoveCompositeData(2);
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the parts in the model.

Arguments

- **Model** (Model)

Model that all parts will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the parts in model m, from 1000000:

```
Part.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged parts in the model.

Arguments

- **Model** (Model)

Model that all the flagged parts will be renumbered in

- **flag** (Flag)

Flag set on the parts that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the parts in model m flagged with f, from 1000000:

```
Part.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select parts using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting parts

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only parts from that model can be selected. If the argument is a Flag then only parts that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any parts can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of parts selected or null if menu cancelled

Return type

Number

Example

To select parts from model m, flagging those selected with flag f, giving the prompt 'Select parts':

```
Part.Select(f, 'Select parts', m);
```

To select parts, flagging those selected with flag f but limiting selection to parts flagged with flag l, giving the prompt 'Select parts':

```
Part.Select(f, 'Select parts', l);
```

SetCompositeData(ipt*[integer]*, mid*[integer]*, thick*[real]*, beta*[real]*, tmid (optinal)*[integer]*, plyid (optional)*[integer]*, shrfac (optional)*[real]*)

Description

Sets the composite data for an integration point in *PART_COMPOSITE.

Arguments

- **ipt** (integer)

The integration point you want to set the data for. **Note that integration points start at 0, not 1**.

- **mid** (integer)

Material ID for the integration point.

- **thick** (real)

Thickness of the integration point.

- **beta** (real)

Material angle of the integration point.

- **tmid (optinal)** (integer)

Thermal material ID for the integration point.

- **plyid (optional)** (integer)

Ply ID for the integration point. This should be used if the _COMPOSITE_LONG option is set for the part.

- **shrfac (optional)** (real)

Transverse shear stress scale factor.

Returns

No return value.

Example

To set the composite data for the 3rd integration point to mat 1, thickness 0.5 and angle 45, for part p:

```
p.SetCompositeData(2, 1, 0.5, 45);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the part.

Arguments

- **flag** (Flag)

Flag to set on the part

Returns

No return value

Example

To set flag f for part p:

```
p.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the part. The part will be sketched until you either call Part.Unsketch(), Part.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the part is sketched. If omitted redraw is true. If you want to sketch several parts

and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch part p:

```
p.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged parts in the model. The parts will be sketched until you either call Part.Unsketch(), Part.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged parts will be sketched in

- **flag** (Flag)

Flag set on the parts that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the parts are sketched. If omitted redraw is true. If you want to sketch flagged parts several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all parts flagged with flag in model m:

```
Part.SketchFlagged(m, flag);
```

TableProperties()

Description

Returns all of the properties available for the part in the part table. The table values are returned in an object. The object property names are the same as the table headers but spaces are replaced with underscore characters and characters other than 0-9, a-z and A-Z are removed to ensure that the property name is valid in JavaScript. If a table value is undefined the property value will be the JavaScript undefined value. If the table value is a valid number it will be a number, otherwise the value will returned as a string.

Arguments

No arguments

Returns

object.

Return type

Object

Example

To get all of the properties for part p:

```
var properties = p.TableProperties();
for (var x in properties)
{
    Message(x+"="+properties[x]);
}
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of parts in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing parts should be counted. If false or omitted referenced but undefined parts will also be included in the total.

Returns

number of parts

Return type

Number

Example

To get the total number of parts in model m:

```
var total = Part.Total(m);
```

Unblank()

Description

Unblanks the part

Arguments

No arguments

Returns

No return value

Example

To unblank part p:

```
p.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the parts in the model.

Arguments

- **Model** (Model)

Model that all parts will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the parts in model m:

```
Part.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged parts in the model.

Arguments

- **Model** (Model)

Model that the flagged parts will be unblanked in

- **flag** (Flag)

Flag set on the parts that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the parts in model m flagged with f:

```
Part.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the parts in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all parts will be unset in

- **flag** (Flag)

Flag to unset on the parts

Returns

No return value

Example

To unset the flag f on all the parts in model m:

```
Part.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the part.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the part is unsketched. If omitted redraw is true. If you want to unsketch several parts and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch part p:

```
p.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all parts.

Arguments

- **Model** (Model)

Model that all parts will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the parts are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all parts in model m:

```
Part.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged parts in the model.

Arguments

- **Model** (Model)

Model that all parts will be unsketched in

- **flag** (Flag)

Flag set on the parts that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the parts are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all parts flagged with flag in model m:

```
Part.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

Part object.

Return type

Part

Example

To check if Part property p.example is a parameter by using the Part.GetParameter() method:

```
if (p.ViewParameters().GetParameter(p.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for part with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description

autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The part object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for part p:

```
p.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for part p:

```
p.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for part. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for part p:

```
p.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this part.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for part p:

```
var xrefs = p.Xrefs();
```

---

toString()

Description

Creates a string containing the part data in keyword format. Note that this contains the keyword header and the keyword cards. See also Part.Keyword() and Part.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for part p in keyword format

```
var str = p.toString();
```

Rigidwall class

The Rigidwall class gives you access to rigidwall cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- FindNodesBehind(flag*[Flag]*)
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- GetRow(row*[integer]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- RemoveRow(row*[integer]*)
- SetFlag(flag*[Flag]*)
- SetRow(row*[integer]*, data*[Array of data]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()

- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

Rigidwall constants

Name	Description
Rigidwall.CYLINDER	Rigidwall is *RIGIDWALL_GEOMETRIC_CYLINDER.
Rigidwall.FLAT	Rigidwall is *RIGIDWALL_GEOMETRIC_FLAT.
Rigidwall.PLANAR	Rigidwall is *RIGIDWALL_PLANAR.
Rigidwall.PRISM	Rigidwall is *RIGIDWALL_GEOMETRIC_PRISM.
Rigidwall.SPHERE	Rigidwall is *RIGIDWALL_GEOMETRIC_SPHERE.

Rigidwall properties

Name	Type	Description
birth	real	Birth time.
boxid	integer	Box for nodes.
d1	real	X component of vector defn.
d2	real	Y component of vector defn.
d3	real	Z component of vector defn.
death	real	Death time.
decaya	real	Friction decay const in local A dir.
decayb	real	Friction decay const in local B dir.
dfrica	real	Dynamic friction coeff in local A dir.
dfricb	real	Dynamic friction coeff in local B dir.
display	logical	DISPLAY flag.
e	real	Young’s modulus of rigidwall (for _DISPLAY option).
exists (read only)	logical	true if rigidwall exists, false if referred to but not defined.
finite	logical	Finite flag.
forces	logical	Forces flag.
fric	real	Friction coefficient.
heading	string	Rigidwall heading
id	logical	true if _ID option is set, false if not
include	integer	The Include file number that the rigidwall is in.
label	integer	Rigidwall number.
lcid	integer	Vel/disp vs time curve number.
lencyl	real	Length of cylinder.
lenl	real	Length of L edge.

lenm	real	Length of M edge.
lenp	real	Length of prism in -ve N.
mass	real	Mass of moving wall.
model (read only)	integer	The Model number that the rigidwall is in.
motion	logical	Motion flag.
moving	logical	Moving flag.
n1	integer	1st node for visualisation.
n2	integer	2nd node for visualisation.
n3	integer	3rd node for visualisation.
n4	integer	4th node for visualisation.
node1	integer	Node 1 for vector defn.
node2	integer	Node 2 for vector defn.
nsegs	integer	Number of subsections.
nsid	integer	Constrained node set included in wall.
nsidex	integer	Constrained node set exempted from wall.
offset	real	Offset for planar option.
opt	integer	Motion type.
ortho	logical	Ortho flag.
pid	integer	Part ID for display of geometric rigidwall (for _DISPLAY option).
pr	real	Poisson’s ratio of rigidwall (for _DISPLAY option).
radcyl	real	Radius of cylinder.
radsph	real	Radius of sphere.
ro	real	Density of rigidwall (for _DISPLAY option).
rwid	integer	Rigidwall number (identical to label).
rwksf	real	Stiffness scaling factor.
sfrica	real	Static friction coeff in local A dir.
sfricb	real	Static friction coeff in local B dir.
soft	integer	No. of cycles to zero relative velocity.
ssid	integer	Segment set number.
type	constant	The rigidwall type. Can be Rigidwall.FLAT, Rigidwall.PRISM, Rigidwall.CYLINDER, Rigidwall.SPHERE, Rigidwall.PLANAR,
v0	real	Initial velocity.
vx	real	X component of motion vector.
vy	real	Y component of motion vector.
vz	real	Z component of motion vector.
wvel	real	Velocity at which nodes weld to wall.
xh	real	Head X coord of outward normal.
xhev	real	Head X coord of edge I vector.
xt	real	Tail X coord of outward normal.

yh	real	Head Y coord of outward normal.
yhev	real	Head Y coord of edge I vector.
yt	real	Tail Y coord of outward normal.
zh	real	Head Z coord of outward normal.
zhev	real	Head Z coord of edge I vector.
zt	real	Tail Z coord of outward normal.

Detailed Description

The Rigidwall class allows you to create, modify, edit rigidwall cards. See the documentation below for more details.

Constructor

new Rigidwall(Model*[Model]*, type*[constant]*, nsid (optional)*[integer]*, rwid (optional)*[integer]*, heading (optional)*[string]*)

Description

Create a new Rigidwall object.

Arguments

- **Model** (Model)

Model that Rigidwall will be created in

- **type** (constant)

Specify the type of rigidwall (Can be Rigidwall.FLAT, Rigidwall.PRISM, Rigidwall.CYLINDER, Rigidwall.SPHERE, Rigidwall.PLANAR)

- **nsid (optional)** (integer)

Node set number.

- **rwid (optional)** (integer)

Rigidwall number

- **heading (optional)** (string)

Title for the Rigidwall

Returns

Rigidwall object

Return type

Rigidwall

Example

To create a new rigidwall 200 of type GEOMETRIC_SPHERE in model m using node set 100 having the title "test wall"

```
var r = new Rigidwall(m, Rigidwall.SPHERE, 200, 100, "test wall");
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a rigidwall.

Arguments

- **Comment** (Comment)

Comment that will be attached to the rigidwall

Returns

No return value

Example

To associate comment c to the rigidwall r:

```
r.AssociateComment(c);
```

Blank()

Description

Blanks the rigidwall

Arguments

No arguments

Returns

No return value

Example

To blank rigidwall r:

```
r.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the rigidwalls in the model.

Arguments

- **Model** (Model)

Model that all rigidwalls will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the rigidwalls in model m:

```
Rigidwall.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged rigidwalls in the model.

Arguments

- **Model** (Model)

Model that all the flagged rigidwalls will be blanked in

- **flag** (Flag)

Flag set on the rigidwalls that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the rigidwalls in model m flagged with f:

```
Rigidwall.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the rigidwall is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if rigidwall r is blanked:

```
if (r.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse rigidwall r:

```
r.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the rigidwall.

Arguments

- **flag** (Flag)

Flag to clear on the rigidwall

Returns

No return value

Example

To clear flag f for rigidwall r:

```
r.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the rigidwall. The target include of the copied rigidwall can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

Rigidwall object

Return type

Rigidwall

Example

To copy rigidwall r into rigidwall z:

```
var z = r.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a rigidwall

Arguments

- **Model** (Model)

Model that the rigidwall will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

Rigidwall object (or null if not made).

Return type

Rigidwall

Example

To start creating a rigidwall in model m:

```
var r = Rigidwall.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a rigidwall.

Arguments

- **Comment** (Comment)

Comment that will be detached from the rigidwall

Returns

No return value

Example

To detach comment c from the rigidwall r:

```
r.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit rigidwall r:

```
r.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for rigidwall with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The rigidwall object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for rigidwall r:

```
r.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for rigidwall r:

```
r.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for rigidwall. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for rigidwall r:

```
r.Error("My custom error");
```

FindNodesBehind(flag*[Flag]*)

Description

Flags nodes that are behind a rigidwall

Arguments

- **flag** (Flag)

Flag to be set on nodes behind rigidwall.

Returns

Number of nodes found

Return type

Number

Example

To set flag f on nodes behind rigidwall w:

```
w.FlagNodesBehind(f);
```

First(Model*[Model]*) [static]

Description

Returns the first rigidwall in the model.

Arguments

- **Model** (Model)

Model to get first rigidwall in

Returns

Rigidwall object (or null if there are no rigidwalls in the model).

Return type

Rigidwall

Example

To get the first rigidwall in model m:

```
var r = Rigidwall.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free rigidwall label in the model. Also see Rigidwall.LastFreeLabel(), Rigidwall.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free rigidwall label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

Rigidwall label.

Return type

Number

Example

To get the first free rigidwall label in model m:

```
var label = Rigidwall.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the rigidwalls in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all rigidwalls will be flagged in

- **flag** (Flag)

Flag to set on the rigidwalls

Returns

No return value

Example

To flag all of the rigidwalls with flag f in model m:

```
Rigidwall.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the rigidwall is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the rigidwall

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if rigidwall r has flag f set on it:

```
if (r.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each rigidwall in the model.
Note that ForEach has been designed to make looping over rigidwalls as fast as possible and so has some limitations.
Firstly, a single temporary Rigidwall object is created and on each function call it is updated with the current rigidwall data. This means that you should not try to store the Rigidwall object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new rigidwalls inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all rigidwalls are in

- **func** (function)

Function to call for each rigidwall

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the rigidwalls in model m:

```
Rigidwall.ForEach(m, test);
function test(r)
{
// r is Rigidwall object
}
```

To call function test for all of the rigidwalls in model m with optional object:

```
var data = { x:0, y:0 };
Rigidwall.ForEach(m, test, data);
function test(r, extra)
{
// r is Rigidwall object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Rigidwall objects or properties for all of the rigidwalls in a model in PRIMER. If the optional property argument is not given then an array of Rigidwall objects is returned. If the property argument is given, that property value for each rigidwall is returned in the array instead of a Rigidwall object

Arguments

- **Model** (Model)

Model to get rigidwalls from

- **property (optional)** (string)

Name for property to get for all rigidwalls in the model

Returns

Array of Rigidwall objects or properties

Return type

Array

Example

To make an array of Rigidwall objects for all of the rigidwalls in model m:

```
var a = Rigidwall.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each rigidwall in model m:

```
var a = Rigidwall.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a rigidwall.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the rigidwall r:

```
var comm_array = r.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of Rigidwall objects for all of the flagged rigidwalls in a model in PRIMER If the optional property argument is not given then an array of Rigidwall objects is returned. If the property argument is given, then that property value for each rigidwall is returned in the array instead of a Rigidwall object

Arguments

- **Model** (Model)

Model to get rigidwalls from

- **flag** (Flag)

Flag set on the rigidwalls that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged rigidwalls in the model

Returns

Array of Rigidwall objects or properties

Return type

Array

Example

To make an array of Rigidwall objects for all of the rigidwalls in model m flagged with f:

```
var r = Rigidwall.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the rigidwalls in model m flagged with f:

```
var a = Rigidwall.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the Rigidwall object for a rigidwall ID.

Arguments

- **Model** (Model)

Model to find the rigidwall in

- **number** (integer)

number of the rigidwall you want the Rigidwall object for

Returns

Rigidwall object (or null if rigidwall does not exist).

Return type

Rigidwall

Example

To get the Rigidwall object for rigidwall 100 in model m

```
var r = Rigidwall.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a Rigidwall property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the Rigidwall.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

rigidwall property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if Rigidwall property r.example is a parameter:

```
Options.property_parameter_names = true;
if (r.GetParameter(r.example) ) do_something...
Options.property_parameter_names = false;
```

To check if Rigidwall property r.example is a parameter by using the GetParameter method:

```
if (r.ViewParameters().GetParameter(r.example) ) do_something...
```

GetRow(row*[integer]*)

Description

Returns the data for an NSEGS card row in the rigidwall.

Arguments

- **row** (integer)

The row you want the data for. **Note row indices start at 0**.

Returns

An array of numbers containing the row variables VL and HEIGHT.

Return type

Number

Example

To get the data for the 2nd row in rigidwall r:

```
var data = r.GetRow(1);
var vl = data[0];
var height = data[1];
```

Keyword()

Description

Returns the keyword for this Rigidwall (*RIGIDWALL). **Note that a carriage return is not added**. See also Rigidwall.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for Rigidwall pm:

```
var key = r.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the Rigidwall. **Note that a carriage return is not added**. See also Rigidwall.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for Rigidwall pm:

```
var cards = r.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last rigidwall in the model.

Arguments

- **Model** (Model)

Model to get last rigidwall in

Returns

Rigidwall object (or null if there are no rigidwalls in the model).

Return type

Rigidwall

Example

To get the last rigidwall in model m:

```
var r = Rigidwall.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free rigidwall label in the model. Also see Rigidwall.FirstFreeLabel(), Rigidwall.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free rigidwall label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

Rigidwall label.

Return type

Number

Example

To get the last free rigidwall label in model m:

```
var label = Rigidwall.LastFreeLabel(m);
```

Next()

Description

Returns the next rigidwall in the model.

Arguments

No arguments

Returns

Rigidwall object (or null if there are no more rigidwalls in the model).

Return type

Rigidwall

Example

To get the rigidwall in model m after rigidwall r:

```
var r = r.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) rigidwall label in the model. Also see Rigidwall.FirstFreeLabel(), Rigidwall.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free rigidwall label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

Rigidwall label.

Return type

Number

Example

To get the next free rigidwall label in model m:

```
var label = Rigidwall.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a rigidwall.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only rigidwalls from that model can be picked. If the argument is a Flag then only rigidwalls that are flagged with *limit* can be selected. If omitted, or null, any rigidwalls from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

Rigidwall object (or null if not picked)

Return type

Rigidwall

Example

To pick a rigidwall from model m giving the prompt 'Pick rigidwall from screen':

```
var r = Rigidwall.Pick('Pick rigidwall from screen', m);
```

Previous()

Description

Returns the previous rigidwall in the model.

Arguments

No arguments

Returns

Rigidwall object (or null if there are no more rigidwalls in the model).

Return type

Rigidwall

Example

To get the rigidwall in model m before rigidwall r:

```
var r = r.Previous();
```

RemoveRow(row*[integer]*)

Description

Removes an NSEGS card row in the *RIGIDWALL.

Arguments

- **row** (integer)

The row you want to remove the data for. **Note that row indices start at 0**.

Returns

No return value.

Example

To remove the second row of data for rigidwall r:

```
r.RemoveRow(1);
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the rigidwalls in the model.

Arguments

- **Model** (Model)

Model that all rigidwalls will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the rigidwalls in model m, from 1000000:

```
Rigidwall.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged rigidwalls in the model.

Arguments

- **Model** (Model)

Model that all the flagged rigidwalls will be renumbered in

- **flag** (Flag)

Flag set on the rigidwalls that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the rigidwalls in model m flagged with f, from 1000000:

```
Rigidwall.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select rigidwalls using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting rigidwalls

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only rigidwalls from that model can be selected. If the argument is a Flag then only rigidwalls that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any rigidwalls can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of rigidwalls selected or null if menu cancelled

Return type

Number

Example

To select rigidwalls from model m, flagging those selected with flag f, giving the prompt 'Select rigidwalls':

```
Rigidwall.Select(f, 'Select rigidwalls', m);
```

To select rigidwalls, flagging those selected with flag f but limiting selection to rigidwalls flagged with flag l, giving the prompt 'Select rigidwalls':

```
Rigidwall.Select(f, 'Select rigidwalls', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the rigidwall.

Arguments

- **flag** (Flag)

Flag to set on the rigidwall

Returns

No return value

Example

To set flag f for rigidwall r:

```
r.SetFlag(f);
```

SetRow(row*[integer]*, data*[Array of data]*)

Description

Sets the data for an NSEGS card row in the *RIGIDWALL.

Arguments

- **row** (integer)

The row you want to set the data for. **Note that row indices start at 0**.

- **data** (Array of data)

The data you want to set the row to

Returns

No return value.

Example

To set the second row of data for rigidwall r to be vl 10.0 and height 1.0:

```
var array = [10.0, 1.0];
r.SetRow(1, array);
```

To append a new row of data (using the same array of values):

```
r.SetRow(r.nsegs, array);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the rigidwall. The rigidwall will be sketched until you either call Rigidwall.Unsketch(), Rigidwall.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the rigidwall is sketched. If omitted redraw is true. If you want to sketch several rigidwalls and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch rigidwall r:

```
r.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged rigidwalls in the model. The rigidwalls will be sketched until you either call Rigidwall.Unsketch(), Rigidwall.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged rigidwalls will be sketched in

- **flag** (Flag)

Flag set on the rigidwalls that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the rigidwalls are sketched. If omitted redraw is true. If you want to sketch flagged rigidwalls several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all rigidwalls flagged with flag in model m:

```
Rigidwall.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of rigidwalls in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing rigidwalls should be counted. If false or omitted referenced but undefined rigidwalls will also be included in the total.

Returns

number of rigidwalls

Return type

Number

Example

To get the total number of rigidwalls in model m:

```
var total = Rigidwall.Total(m);
```

Unblank()

Description

Unblanks the rigidwall

Arguments

No arguments

Returns

No return value

Example

To unblank rigidwall r:

```
r.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the rigidwalls in the model.

Arguments

- **Model** (Model)

Model that all rigidwalls will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the rigidwalls in model m:

```
Rigidwall.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged rigidwalls in the model.

Arguments

- **Model** (Model)

Model that the flagged rigidwalls will be unblanked in

- **flag** (Flag)

Flag set on the rigidwalls that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the rigidwalls in model m flagged with f:

```
Rigidwall.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the rigidwalls in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all rigidwalls will be unset in

- **flag** (Flag)

Flag to unset on the rigidwalls

Returns

No return value

Example

To unset the flag f on all the rigidwalls in model m:

```
Rigidwall.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the rigidwall.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the rigidwall is unsketched. If omitted redraw is true. If you want to unsketch several rigidwalls and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch rigidwall r:

```
r.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all rigidwalls.

Arguments

- **Model** (Model)

Model that all rigidwalls will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the rigidwalls are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all rigidwalls in model m:

```
Rigidwall.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged rigidwalls in the model.

Arguments

- **Model** (Model)

Model that all rigidwalls will be unsketched in

- **flag** (Flag)

Flag set on the rigidwalls that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the rigidwalls are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all rigidwalls flagged with flag in model m:

```
Rigidwall.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

Rigidwall object.

Return type

Rigidwall

Example

To check if Rigidwall property r.example is a parameter by using the Rigidwall.GetParameter() method:

```
if (r.ViewParameters().GetParameter(r.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for rigidwall with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The rigidwall object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for rigidwall r:

```
r.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for rigidwall r:

```
r.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for rigidwall. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for rigidwall r:

```
r.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this rigidwall.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for rigidwall r:

```
var xrefs = r.Xrefs();
```

---

toString()

Description

Creates a string containing the Rigidwall data in keyword format. Note that this contains the keyword header and the keyword cards. See also Rigidwall.Keyword() and Rigidwall.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for Rigidwall pm in keyword format

```
var r = r.toString();
```

Section class

The Section class gives you access to section cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- ExtractColour()
- Flagged(flag*[Flag]*)
- GetBetaData(ipt*[integer]*)
- GetComments()
- GetLmcData(i*[integer]*)
- GetParameter(prop*[string]*)
- GetPointData(ipt*[integer]*)
- GetUserData(ipt*[integer]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetBetaData(ipt*[integer]*, beta*[real]*)
- SetFlag(flag*[Flag]*)

- SetLmcData(ipt*[integer]*, lmc*[real]*)
- SetPointData(ipt*[integer]*, nodeid*[integer]*, vecid*[integer]*, area*[real]*)
- SetUserData(ipt*[integer]*, xi*[real]*, eta*[real]*, zeta (SOLID) **or**
  wgt (SHELL)*[real]*, wgt (SOLID only)*[real]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

Section constants

Name	Description
Section.ALE1D	Section ale1d type
Section.ALE2D	Section ale2d type
Section.BEAM	Section beam type
Section.DISCRETE	Section discrete type
Section.IGA_SHELL	Section IGA shell type
Section.IGA_SOLID	Section IGA solid type
Section.POINT_SOURCE	Section point source type
Section.SEATBELT	Section seatbelt type
Section.SHELL	Section shell type
Section.SOLID	Section solid type
Section.SPH	Section sph type
Section.TSHELL	Section thick shell type

Section properties

Name	Type	Description
aafac	real	ALE advection factor(SHELL, SOLID)
aet	integer	Ambient element type (ALE1D, ALE2D, SOLID)
afac	real	Smoothing weight factor - Simple average (SHELL, SOLID)
ale	logical	If _ALE option is set. Can be true or false (SHELL, SOLID)
aleform	integer	ALE formulation (ALE1D, ALE2D)
baselm	integer	Base element type for XFEM (SHELL)
bfac	real	Smoothing weight factor - Volume weighting (SHELL, SOLID)
cfac	real	Smoothing weight factor - Isoparametric (SHELL, SOLID)
cmid	integer	Cohesive material (SHELL, SOLID)
cohthk	real	Cohesive thickness (SOLID)
colour	Colour	The colour of the section
dfac	real	Smoothing weight factor - Equipotential (SHELL, SOLID)
domint	integer	Domain integration in XFEM (SHELL)
dr	real	PERI normalized horizon size (SOLID)

dx	real	Normalized dilation parameter of kernel function in X (SHELL, SOLID)
dy	real	Normalized dilation parameter of kernel function in Y (SHELL, SOLID)
dz	real	Normalized dilation parameter of kernel function in Z (SOLID)
efac	real	Smoothing weight factor - Equilibrium (SHELL)
efg	logical	If _EFG option is set. Can be true or false (SHELL, SOLID)
elform	integer	Element formulation (ALE1D, ALE2D, BEAM, IGA_SHELL, IGA_SOLID, SHELL, SOLID, TSHELL)
end	real	End time for smoothing (SHELL, SOLID)
exists (read only)	logical	true if section exists, false if referred to but not defined.
failcr	integer	Different failure criteria (SHELL)
fs	real	SPG Failure strain if IDAM = 1 (SOLID)
icomp	integer	Composite flag (IGA_SHELL, IGA_SOLID, SHELL, TSHELL)
idam	integer	SPG Option of damage mechanism (SOLID)
idila	integer	Normalized dilation parameter definition (SOLID)
idim	integer	Domain integration method (SOLID)
iebt	integer	Essential boundary condition treatment (SOLID)
ihgf	integer	Flag for using hourglass stabilization (SHELL, SOLID)
iloc	integer	Coordinate system option (SHELL)
include	integer	The Include file number that the section is in.
ispline	integer	EFG kernel function definition (SHELL, SOLID)
itaj	integer	Flag for setting up finite element matrices (SHELL, SOLID)
itb	integer	SPG Stabilization flag (SOLID)
ithelfm	integer	THERMAL shell formulation (SHELL)
iunf	integer	Flag for using nodal fibre vectors (SHELL)
kernel	integer	SPG kernel type approximation (SOLID)
label	integer or string	Section ID (all types) or character label. Also see the secid property which is an alternative name for this.
lmc	integer	Number of property parameters (SHELL, SOLID)
lprint	integer	Debug printout option (SHELL)
lscale	real	SPG length scale for displacement regularisation (SOLID)
misc	logical	If _MISC option is set. Can be true or false (SHELL, SOLID)
model (read only)	integer	The Model number that the section is in.
nhsv	integer	Number of history variables (SHELL, SOLID)
nip	integer	Number of integration points (IGA_SHELL, IGA_SOLID, SHELL, SOLID, TSHELL)
nipp	integer	Number of in-plane integration points (SHELL)
nxdof	integer	Number of extra degrees of freedom per node (SHELL, SOLID)
peri	logical	If _PERI option is set. Can be true or false (SOLID)
propcr	integer	Not used (SHELL)

propt	real	Printout option (SHELL, TSHELL)
ptype	integer	PERI peridynamics formulation (SOLID)
qr	real	Quadrature rule (BEAM, IGA_SHELL, IGA_SOLID, SHELL, TSHELL)
secid	integer or string	Section ID (all types) or character label. Also see the label property which is an alternative name for this.
shrf	real	Shear correction factor (BEAM, IGA_SHELL, IGA_SOLID, SHELL, TSHELL)
smstep	integer	SPG Interval of timestep to conduction displ regularisation (SOLID)
spg	logical	If _SPG option is set. Can be true or false (SOLID)
start	real	Time imposed SPH approximation is activated (SPH) **or** Start time for smoothing (SHELL, SOLID)
stretch	real	SPG stretching parameter if IDAM = 1 (SOLID)
swtime	real	SPG Time to switch from updated Lagrangian to Eulerian kernel (SOLID)
thermal	logical	If _THERMAL option is set. Can be true or false (SHELL)
thick	real	Thickness (ALE1D, SEATBELT)
title	string	Section title (all types)
toldef	real	Deformation tolerance (SOLID)
transparency	integer	The transparency of the section (0-100) 0% is opaque, 100% is transparent.
type (read only)	constant	Section type. Can be Section.ALE1D, Section.ALE2D, Section.BEAM, Section.DISCRETE, Section.IGA_SHELL, Section.IGA_SOLID, Section.POINT_SOURCE, Section.SEATBELT, Section.SHELL, Section.SOLID, Section.SPH or Section.TSHELL
xfem	logical	If _THERMAL option is set. Can be true or false (SHELL)

Properties for BEAM

Name	Type	Description
a	real	Cross sectional area
aisc	logical	If _AISC option is set. Can be true or false
aisc_label	string	AISC section label
ca	real	Cable area
cid	integer	Coordinate system ID for orientation
cst	real	Cross section type
d1	real	Input parameter 1 for section type
d2	real	Input parameter 2 for section type
d3	real	Input parameter 3 for section type
d4	real	Input parameter 4 for section type
d5	real	Input parameter 5 for section type
d6	real	Input parameter 6 for section type
dofn1	real	Active degree of freedom at node 1
dofn2	real	Active degree of freedom at node 2
iner	real	Mass moment of inertia
iovpr	integer	Print flag for the elbow ovalization degrees of freedom (elform 14)

iprstr	integer	Flag for adding stress due to pressure into the material routine (elform 14)
irr	real	Irr
iss	real	Iss
ist	real	Ist
itoff	real	Option to specify torsional behaviour for spotwelds
itorm	real	Itorm
itt	real	Itt
iw	real	Warping constant
iwr	real	Warping constant
iyr	real	IYR integral
izr	real	IZR integral
j	real	torsional constant
nsloc	real	Location of s reference surface
nsm	real	Non structural mass per unit length
ntloc	real	Location of t reference surface
offset	real	Offset for cable
pr	real	Pressure inside elements (elform 14)
print	real	Output spot force resultants from spotwelds
rampt	real	Ramp up time for dynamic relaxation
rrcon	real	r rotational constraint
sa	real	Shear area
sat	real	Perpendicular shear area
scoor	real	Location of triad for discrete beam
srcon	real	s rotational constraint
stress	real	Initial stress for dynamic relaxation
stype	string	Section type
trcon	real	t rotational constraint
ts1	real	s thickness or outer diameter at N1
ts2	real	s thickness or outer diameter at N2
tt1	real	t thickness or inner diameter at N1
tt2	real	t thickness or inner diameter at N2
vol	real	Volume of discrete beam
ys	real	s coordinate of shear centre of cross section
zs	real	t coordinate of shear centre of cross section

Properties for DISCRETE

Name	**Type**	**Description**
cdl	real	Deflection limit in compression
cl	real	Clearance
dro	integer	Displacement/rotation option
fd	real	Failure deflection

kd	real	Dynamic magnification factor
tdl	real	Deflection limit in tension
v0	real	Test velocity

Properties for IGA_SHELL

Name	Type	Description
irl	integer	Lamina integration rule
nloc	real	Location of reference surface
t	real	Shell thickness

Properties for IGA_SOLID

Name	Type	Description
ir	integer	Integration rule

Properties for POINT SOURCE

Name	Type	Description
lcidt	integer	Temperature loadcurve ID
lcidvel	integer	Inlet flow velocity loadcurve ID
lcidvolr	integer	Relative volume loadcurve ID
lcmdot1	integer	Mass flowrate loadcurve for gas 1
lcmdot2	integer	Mass flowrate loadcurve for gas 2
lcmdot3	integer	Mass flowrate loadcurve for gas 3
lcmdot4	integer	Mass flowrate loadcurve for gas 4
lcmdot5	integer	Mass flowrate loadcurve for gas 5
lcmdot6	integer	Mass flowrate loadcurve for gas 6
lcmdot7	integer	Mass flowrate loadcurve for gas 7
lcmdot8	integer	Mass flowrate loadcurve for gas 8
mixture	logical	If _MIXTURE option is set. Can be true or false
nidlc001	integer	1st node ID defining a local coordinate
nidlc002	integer	2nd node ID defining a local coordinate
nidlc003	integer	3rd node ID defining a local coordinate
points	integer	Number of point sources

Properties for SEATBELT

Name	Type	Description
area	real	Optional cross sectional area used in contact

Properties for SHELL

Name	Type	Description

edgset	integer	Edge node set
idof	real	Thickness field value
marea	real	Non structural mass per unit area
nloc	integer	Location of reference surface
setyp	integer	2D solid element type
t1	real	Thickness at Node 1
t2	real	Thickness at Node 2
t3	real	Thickness at Node 3
t4	real	Thickness at Node 4

Properties for SOLID

Name	Type	Description
cohoff	real	Relative location of cohesive layer (for cohesive solid elements 20 and 22)
ds	real	Displacement jump
ecut	real	Minimum distance to the node that a crack surface can cut to the edge
gaskett	real	Gasket thickness for converting elform 19-22 elements to gasket element
ibr	integer	Branching
iken	integer	approximation
ips	integer	Pressure smoothing/recovery
sf	real	Failure strain
stime	real	Time to switch from stabilized EFG to standard EFG formulation

Properties for SPH

Name	Type	Description
cslh	real	Smoothing length constant
death	real	Time imposed SPH approximation is stopped
ellipse	logical	If _ELLIPSE option is set (was _TENSOR pre R8). Can be true or false
hmax	real	Max smoothing length scale factor
hmin	real	Min smoothing length scale factor
hxcslh	real	Constant for smoothing length in X for tensor/ellipse case
hxini	real	Initial smoothing length in X for tensor/ellipse case
hycslh	real	Constant for smoothing length in Y for tensor/ellipse case
hyini	real	Initial smoothing length in Y for tensor/ellipse case
hzcslh	real	Constant for smoothing length in Z for tensor/ellipse case
hzini	real	Initial smoothing length in Z for tensor/ellipse case
iform	integer	SPH element formulation
interaction	logical	If _INTERACTION option is set. Can be true or false
sphini	real	Optional initial smoothing length
tensor	logical	If _TENSOR option is set (_ELLIPSE from R8 onwards). Can be true or false

user	logical	If _USER option is set. Can be true or false

Properties for TSHELL

Name	Type	Description
tshear	integer	Flag for transverse shear strain or stress distribution

Detailed Description

The Section class allows you to create, modify, edit and manipulate section cards. See the documentation below for more details.

Constructor

new Section(Model*[Model]*, secid*[integer or string]*, type*[constant]*, title (optional)*[string]*)

Description

Create a new Section object.

Arguments

- **Model** (Model)

Model that section will be created in

- **secid** (integer or string)

Section number or character label

- **type** (constant)

Section type. Can be Section.BEAM, Section.DISCRETE, Section.IGA_SHELL, Section.IGA_SOLID, Section.POINT_SOURCE, Section.SEATBELT, Section.SHELL, Section.SOLID, Section.SPH or Section.TSHELL

- **title (optional)** (string)

Title for the section

Returns

Section object

Return type

Section

Example

To create a new section, type shell, called 'Example' in model m with label 100:

```
var s = new Section(m, 100, Section.SHELL, 'Example');
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a section.

Arguments

- **Comment** (Comment)

Comment that will be attached to the section

Returns

No return value

Example

To associate comment c to the section s:

```
s.AssociateComment(c);
```

Blank()

Description

Blanks the section

Arguments

No arguments

Returns

No return value

Example

To blank section s:

```
s.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the sections in the model.

Arguments

- **Model** (Model)

Model that all sections will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the sections in model m:

```
Section.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged sections in the model.

Arguments

- **Model** (Model)

Model that all the flagged sections will be blanked in

- **flag** (Flag)

Flag set on the sections that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the sections in model m flagged with f:

```
Section.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the section is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if section s is blanked:

```
if (s.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse section s:

```
s.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the section.

Arguments

- **flag** (Flag)

Flag to clear on the section

Returns

No return value

Example

To clear flag f for section s:

```
s.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the section. The target include of the copied section can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

Section object

Return type

Section

Example

To copy section s into section z:

```
var z = s.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a section

Arguments

- **Model** (Model)

Model that the section will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

Section object (or null if not made).

Return type

Section

Example

To start creating a section in model m:

```
var s = Section.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a section.

Arguments

- **Comment** (Comment)

Comment that will be detached from the section

Returns

No return value

Example

To detach comment c from the section s:

```
s.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit section s:

```
s.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for section with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The section object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for section s:

```
s.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for section s:

```
s.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for section. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for section s:

```
s.Error("My custom error");
```

ExtractColour()

Description

Extracts the **actual** colour used for section.
By default in PRIMER many entities such as elements get their colour automatically from the part that they are in. PRIMER cycles through 13 default colours based on the label of the entity. In this case the section colour property will return the value Colour.PART instead of the actual colour. This method will return the actual colour which is used for drawing the section.

Arguments

No arguments

Returns

colour value (integer)

Return type

Number

Example

To return the colour used for drawing section s:

```
var colour = s.ExtractColour();
```

First(Model*[Model]*) [static]

Description

Returns the first section in the model.

Arguments

- **Model** (Model)

Model to get first section in

Returns

Section object (or null if there are no sections in the model).

Return type

Section

Example

To get the first section in model m:

```
var s = Section.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free section label in the model. Also see Section.LastFreeLabel(), Section.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free section label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

Section label.

Return type

Number

Example

To get the first free section label in model m:

```
var label = Section.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the sections in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all sections will be flagged in

- **flag** (Flag)

Flag to set on the sections

Returns

No return value

Example

To flag all of the sections with flag f in model m:

```
Section.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the section is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the section

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if section s has flag f set on it:

```
if (s.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each section in the model.
Note that ForEach has been designed to make looping over sections as fast as possible and so has some limitations.
Firstly, a single temporary Section object is created and on each function call it is updated with the current section data. This means that you should not try to store the Section object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new sections inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all sections are in

- **func** (function)

Function to call for each section

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the sections in model m:

```
Section.ForEach(m, test);
function test(s)
{
// s is Section object
}
```

To call function test for all of the sections in model m with optional object:

```
var data = { x:0, y:0 };
Section.ForEach(m, test, data);
function test(s, extra)
{
// s is Section object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Section objects or properties for all of the sections in a model in PRIMER. If the optional property argument is not given then an array of Section objects is returned. If the property argument is given, that property value for each section is returned in the array instead of a Section object

Arguments

- **Model** (Model)

Model to get sections from

- **property (optional)** (string)

Name for property to get for all sections in the model

Returns

Array of Section objects or properties

Return type

Array

Example

To make an array of Section objects for all of the sections in model m:

```
var a = Section.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each section in model m:

```
var a = Section.GetAll(m, 'foo');
```

GetBetaData(ipt*[integer]*)

Description

Returns the beta angle data for an integration point in *SECTION_IGA_SHELL, *SECTION_SHELL or *SECTION_TSHELL.

Arguments

- **ipt** (integer)

The integration point you want the data for. **Note that integration points start at 0, not 1**.

Returns

real

Return type

Number

Example

To get the beta angle for the 3rd integration point for section shell s:

```
if (s.icomp && s.nip >= 3)
{
    var beta = s.GetBetaData(2);
}
```

GetComments()

Description

Extracts the comments associated to a section.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the section s:

```
var comm_array = s.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of Section objects for all of the flagged sections in a model in PRIMER If the optional property argument is not given then an array of Section objects is returned. If the property argument is given, then that property value for each section is returned in the array instead of a Section object

Arguments

- **Model** (Model)

Model to get sections from

- **flag** (Flag)

Flag set on the sections that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged sections in the model

Returns

Array of Section objects or properties

Return type

Array

Example

To make an array of Section objects for all of the sections in model m flagged with f:

```
var s = Section.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the sections in model m flagged with f:

```
var a = Section.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the Section object for a section ID.

Arguments

- **Model** (Model)

Model to find the section in

- **number** (integer)

number of the section you want the Section object for

Returns

Section object (or null if section does not exist).

Return type

Section

Example

To get the Section object for section 100 in model m

```
var s = Section.GetFromID(m, 100);
```

GetLmcData(i*[integer]*)

Description

Returns the LMC property parameter for *SECTION_SHELL or *SECTION_SOLID.

Arguments

- **i** (integer)

The point you want the parameter for. **Note that points start at 0, not 1**.

Returns

real

Return type

Number

Example

To get the 3rd LMC parameter for section shell s:

```
if (s.lmc >= 3)
{
    var p = s.GetLmcData(2);
}
```

GetParameter(prop*[string]*)

Description

Checks if a Section property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the Section.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

section property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if Section property s.example is a parameter:

```
Options.property_parameter_names = true;
if (s.GetParameter(s.example) ) do_something...
Options.property_parameter_names = false;
```

To check if Section property s.example is a parameter by using the GetParameter method:

```
if (s.ViewParameters().GetParameter(s.example) ) do_something...
```

GetPointData(ipt*[integer]*)

Description

Returns the point data for a single point in *SECTION_POINT_SOURCE.

Arguments

- **ipt** (integer)

The point you want the data for. **Note that integration points start at 0, not 1**.

Returns

An array of numbers containing the node id, vector id and orifice area.

Return type

Number

Example

To get the data for the 3rd point for section point source s:

```
if (s.points >= 3)
{
    var pt_data = s.GetPointData(3);
}
```

GetUserData(ipt*[integer]*)

Description

Returns the user defined data for an integration point in *SECTION_SHELL and *SECTION_SOLID.

Arguments

- **ipt** (integer)

The integration point you want the data for. **Note that integration points start at 0, not 1**.

Returns

An array containing the data (XI, ETA, WGT for *SECTION_SHELL, XI, ETA, ZETA, WGT for *SECTION_SOLID).

Return type

Array

Example

To get the data for the 3rd integration point for section shell s:

```
if (s.nipp >= 3)
{
    var user_data = s.GetUserData(2);
}
```

Keyword()

Description

Returns the keyword for this section (*SECT, *SECT_SCALAR or *SECT_SCALAR_VALUE). **Note that a carriage return is not added**. See also Section.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for section s:

```
var key = s.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the section. **Note that a carriage return is not added**. See also Section.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for section s:

```
var cards = n.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last section in the model.

Arguments

- **Model** (Model)

Model to get last section in

Returns

Section object (or null if there are no sections in the model).

Return type

Section

Example

To get the last section in model m:

```
var s = Section.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free section label in the model. Also see Section.FirstFreeLabel(), Section.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free section label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

Section label.

Return type

Number

Example

To get the last free section label in model m:

```
var label = Section.LastFreeLabel(m);
```

Next()

Description

Returns the next section in the model.

Arguments

No arguments

Returns

Section object (or null if there are no more sections in the model).

Return type

Section

Example

To get the section in model m after section s:

```
var s = s.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) section label in the model. Also see Section.FirstFreeLabel(), Section.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free section label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

Section label.

Return type

Number

Example

To get the next free section label in model m:

```
var label = Section.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a section.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only sections from that model can be picked. If the argument is a Flag then only sections that are flagged with *limit* can be selected. If omitted, or null, any sections from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

Section object (or null if not picked)

Return type

Section

Example

To pick a section from model m giving the prompt 'Pick section from screen':

```
var s = Section.Pick('Pick section from screen', m);
```

Previous()

Description

Returns the previous section in the model.

Arguments

No arguments

Returns

Section object (or null if there are no more sections in the model).

Return type

Section

Example

To get the section in model m before section s:

```
var s = s.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the sections in the model.

Arguments

- **Model** (Model)

Model that all sections will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the sections in model m, from 1000000:

```
Section.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged sections in the model.

Arguments

- **Model** (Model)

Model that all the flagged sections will be renumbered in

- **flag** (Flag)

Flag set on the sections that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the sections in model m flagged with f, from 1000000:

```
Section.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select sections using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting sections

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only sections from that model can be selected. If the argument is a Flag then only sections that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any sections can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of sections selected or null if menu cancelled

Return type

Number

Example

To select sections from model m, flagging those selected with flag f, giving the prompt 'Select sections':

```
Section.Select(f, 'Select sections', m);
```

To select sections, flagging those selected with flag f but limiting selection to sections flagged with flag l, giving the prompt 'Select sections':

```
Section.Select(f, 'Select sections', l);
```

SetBetaData(ipt*[integer]*, beta*[real]*)

Description

Sets the beta angle for an integration point in *SECTION_IGA_SHELL, *SECTION_SHELL or *SECTION_TSHELL.

Arguments

- **ipt** (integer)

The integration point you want to set the data for. **Note that integration points start at 0, not 1**.

- **beta** (real)

Beta angle for the integration point.

Returns

No return value.

Example

To set the beta angle for the 3rd integration point to 45, for section s:

```
s.SetBetaData(2, 45);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the section.

Arguments

- **flag** (Flag)

Flag to set on the section

Returns

No return value

Example

To set flag f for section s:

```
s.SetFlag(f);
```

SetLmcData(ipt*[integer]*, lmc*[real]*)

Description

Sets the lmc parameter for a point in *SECTION_SHELL or *SECTION_SOLID.

Arguments

- **ipt** (integer)

The point you want to set the data for. **Note that points start at 0, not 1**.

- **lmc** (real)

Lmc parameter for the point.

Returns

No return value.

Example

To set the 3rd lmc point to 0.1, for section s:

```
s.SetLmcData(2, 0.1);
```

SetPointData(ipt*[integer]*, nodeid*[integer]*, vecid*[integer]*, area*[real]*)

Description

Sets the data for a single point in *SECTION_POINT_SOURCE.

Arguments

- **ipt** (integer)

The point you want to set the data for. **Note that integration points start at 0, not 1**.

- **nodeid** (integer)

Node ID for the point.

- **vecid** (integer)

Vector ID for the point.

- **area** (real)

Orifice area for the point.

Returns

No return value.

Example

To set the data for the 3rd point to node 1, vector 10 and area 0.2, for section s:

```
s.SetPointData(2, 1, 10, 0.2);
```

SetUserData(ipt*[integer]*, xi*[real]*, eta*[real]*, zeta (SOLID) **or** wgt (SHELL)*[real]*, wgt (SOLID only)*[real]*)

Description

Sets the user defined data for an integration point in *SECTION_SHELL and *SECTION_SOLID.

Arguments

- **ipt** (integer)

The integration point you want to set the data for. **Note that integration points start at 0, not 1**.

- **xi** (real)

First isoparametric coordinate.

- **eta** (real)

Second isoparametric coordinate.

- **zeta (SOLID) or**
  wgt (SHELL) (real)

Second isoparametric coordinate (SOLID) **or**
Isoparametric weight (SHELL)

- **wgt (SOLID only)** (real)

Isoparametric weight (SOLID)

Returns

No return value.

Example

To set the user data for the 3rd integration point to xi 0.5, eta 0.5, zeta -0.5, wgt 0.125, for section solid s:

```
s.SetUserData(2, 0.5, 0.5, -0.5, 0.125);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the section. The section will be sketched until you either call Section.Unsketch(), Section.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the section is sketched. If omitted redraw is true. If you want to sketch several sections and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch section s:

```
s.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged sections in the model. The sections will be sketched until you either call Section.Unsketch(), Section.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged sections will be sketched in

- **flag** (Flag)

Flag set on the sections that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the sections are sketched. If omitted redraw is true. If you want to sketch flagged sections several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all sections flagged with flag in model m:

```
Section.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of sections in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing sections should be counted. If false or omitted referenced but undefined sections will also be included in the total.

Returns

number of sections

Return type

Number

Example

To get the total number of sections in model m:

```
var total = Section.Total(m);
```

Unblank()

Description

Unblanks the section

Arguments

No arguments

Returns

No return value

Example

To unblank section s:

```
s.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the sections in the model.

Arguments

- **Model** (Model)

Model that all sections will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the sections in model m:

```
Section.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged sections in the model.

Arguments

- **Model** (Model)

Model that the flagged sections will be unblanked in

- **flag** (Flag)

Flag set on the sections that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the sections in model m flagged with f:

```
Section.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the sections in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all sections will be unset in

- **flag** (Flag)

Flag to unset on the sections

Returns

No return value

Example

To unset the flag f on all the sections in model m:

```
Section.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the section.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the section is unsketched. If omitted redraw is true. If you want to unsketch several sections and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch section s:

```
s.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all sections.

Arguments

- **Model** (Model)

Model that all sections will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the sections are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all sections in model m:

```
Section.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged sections in the model.

Arguments

- **Model** (Model)

Model that all sections will be unsketched in

- **flag** (Flag)

Flag set on the sections that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the sections are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all sections flagged with flag in model m:

```
Section.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

Section object.

Return type

Section

Example

To check if Section property s.example is a parameter by using the Section.GetParameter() method:

```
if (s.ViewParameters().GetParameter(s.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for section with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description

autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The section object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for section s:

```
s.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for section s:

```
s.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for section. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for section s:

```
s.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this section.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for section s:

```
var xrefs = s.Xrefs();
```

---

toString()

Description

Creates a string containing the section data in keyword format. Note that this contains the keyword header and the keyword cards. See also Section.Keyword() and Section.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for section s in keyword format

```
var str = s.toString();
```

SensorControl class

The SensorControl class gives you access to *SENSOR_CONTROL keyword in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

SensorControl properties

Name	Type	Description

cntlid	integer	SensorControl number. The label property is an alternative name for this.
defcv	real/string	Default curve value when a curve is not active for TYPE = CURVE only. If DEFCRV = "LASTSTEP", the curve value right before the curve is turned off becomes the default curve value.
estyp	string	Element Set Type to be controlled. Can be "BEAM", "DISC", "SHELL", "SOLID", "TSHELL".
exists (read only)	logical	true if *SENSOR_CONTROL exists, false if referred to but not defined.
include	integer	The Include file number that the *SENSOR_CONTROL is in.
initstt	string	Initial status. Can be "On" or "Off".
label	integer	SensorControl number. The cntlid property is an alternative name for this.
model (read only)	integer	The Model number that the *SENSOR_CONTROL is in.
nrep	integer	Number of repeat of cycle of switches.
swit1	integer	ID of 1st switch.
swit2	integer	ID of 2nd switch.
swit3	integer	ID of 3rd switch.
swit4	integer	ID of 4th switch.
swit5	integer	ID of 5th switch.
swit6	integer	ID of 6th switch.
swit7	integer	ID of 7th switch.
timeoff	integer	Flag for offset of time in curve.
timeoff_idiscl	integer	Flag for offset of time in curve or flag for the reference length of the discrete element
type	string	Entity to be controlled. Can be "AIRBAG", "BAGVENTPOP", "BELTPRET", "BELTRETRA", "BELTSLIP", "CONTACT", "CONTACT2D", "CNRB", "DEF2RIG", "DISC-ELE", "DISC-ELES", "ELESET", "FUNCTION", "JOINT", "JOINTSTIFF", "LOADTHM", "M PRESSURE", "RWALL", "SPC", "SPOTWELD".
typeid	integer	ID of entity to be controlled if type is not FUNCTION or input value for FUNCTION.

Detailed Description

The SensorControl class allows you to create, modify, edit and manipulate *SENSOR_CONTROL. See the documentation below for more details.

Constructor

new SensorControl(Model*[Model]*, Sensor control ID*[integer]*, Type*[string]*, Type ID (optional)*[integer]*, estyp (optional)*[string]*)

Description

Create a new SensorControl object.

Arguments

- **Model** (Model)

Model that *SENSOR_CONTROL will be created in

- **Sensor control ID** (integer)

SensorControl id.

- **Type** (string)

Entity type to be controlled. Can be "AIRBAG", "BAGVENTPOP", "BELTPRET", "BELTRETRA", "BELTSLIP", "CONTACT", "CONTACT2D", "DEF2RIG", "CURVE", "DISC-ELE", "DISC-ELES", "ELESET", "FUNCTION", "JOINT", "JOINTSTIFF", "M PRESSURE", "RWALL", "SPC", "SPOTWELD".

- **Type ID (optional)** (integer)

ID of entity to be controlled if type is not FUNCTION or input value for FUNCTION.

- **estyp (optional)** (string)

Element Set Type to be controlled. Can be "BEAM", "DISC", "SHELL", "SOLID", "TSHELL". **Required only if Type argument is "ELESET"**.

Returns

SensorControl object

Return type

SensorControl

Example

To create a new *SENSOR_CONTROL in model m with label 100 and type JOINT:

```
var sc = new SensorControl(m, 100, "JOINT");
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a *SENSOR_CONTROL.

Arguments

- **Comment** (Comment)

Comment that will be attached to the *SENSOR_CONTROL

Returns

No return value

Example

To associate comment c to the *SENSOR_CONTROL sc:

```
sc.AssociateComment(c);
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse *SENSOR_CONTROL sc:

```
sc.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the *SENSOR_CONTROL.

Arguments

- **flag** (Flag)

Flag to clear on the *SENSOR_CONTROL

Returns

No return value

Example

To clear flag f for *SENSOR_CONTROL sc:

```
sc.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the *SENSOR_CONTROL. The target include of the copied *SENSOR_CONTROL can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

SensorControl object

Return type

SensorControl

Example

To copy *SENSOR_CONTROL sc into *SENSOR_CONTROL z:

```
var z = sc.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a *SENSOR_CONTROL

Arguments

- **Model** (Model)

Model that the *SENSOR_CONTROL will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

SensorControl object (or null if not made).

Return type

SensorControl

Example

To start creating a *SENSOR_CONTROL in model m:

```
var sc = SensorControl.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a *SENSOR_CONTROL.

Arguments

- **Comment** (Comment)

Comment that will be detached from the *SENSOR_CONTROL

Returns

No return value

Example

To detach comment c from the *SENSOR_CONTROL sc:

```
sc.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit *SENSOR_CONTROL sc:

```
sc.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for *SENSOR_CONTROL with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The *SENSOR_CONTROL object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for *SENSOR_CONTROL sc:

```
sc.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for *SENSOR_CONTROL sc:

```
sc.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for *SENSOR_CONTROL. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for *SENSOR_CONTROL sc:

```
sc.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first *SENSOR_CONTROL in the model.

Arguments

- **Model** (Model)

Model to get first *SENSOR_CONTROL in

Returns

SensorControl object (or null if there are no *SENSOR_CONTROLs in the model).

Return type

SensorControl

Example

To get the first *SENSOR_CONTROL in model m:

```
var sc = SensorControl.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free *SENSOR_CONTROL label in the model. Also see SensorControl.LastFreeLabel(), SensorControl.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free *SENSOR_CONTROL label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

SensorControl label.

Return type

Number

Example

To get the first free *SENSOR_CONTROL label in model m:

```
var label = SensorControl.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the *SENSOR_CONTROLs in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all *SENSOR_CONTROLs will be flagged in

- **flag** (Flag)

Flag to set on the *SENSOR_CONTROLs

Returns

No return value

Example

To flag all of the *SENSOR_CONTROLs with flag f in model m:

```
SensorControl.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the *SENSOR_CONTROL is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the *SENSOR_CONTROL

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if *SENSOR_CONTROL sc has flag f set on it:

```
if (sc.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each *SENSOR_CONTROL in the model.
Note that ForEach has been designed to make looping over *SENSOR_CONTROLs as fast as possible and so has some limitations.
Firstly, a single temporary SensorControl object is created and on each function call it is updated with the current *SENSOR_CONTROL data. This means that you should not try to store the SensorControl object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new *SENSOR_CONTROLs inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all *SENSOR_CONTROLs are in

- **func** (function)

Function to call for each *SENSOR_CONTROL

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the *SENSOR_CONTROLs in model m:

```
SensorControl.ForEach(m, test);
function test(sc)
{
// sc is SensorControl object
}
```

To call function test for all of the *SENSOR_CONTROLs in model m with optional object:

```
var data = { x:0, y:0 };
SensorControl.ForEach(m, test, data);
function test(sc, extra)
{
// sc is SensorControl object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of SensorControl objects or properties for all of the *SENSOR_CONTROLs in a model in PRIMER. If the optional property argument is not given then an array of SensorControl objects is returned. If the property argument is given, that property value for each *SENSOR_CONTROL is returned in the array instead of a SensorControl object

Arguments

- **Model** (Model)

Model to get *SENSOR_CONTROLs from

- **property (optional)** (string)

Name for property to get for all *SENSOR_CONTROLs in the model

Returns

Array of SensorControl objects or properties

Return type

Array

Example

To make an array of SensorControl objects for all of the *SENSOR_CONTROLs in model m:

```
var a = SensorControl.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each *SENSOR_CONTROL in model m:

```
var a = SensorControl.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a *SENSOR_CONTROL.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the *SENSOR_CONTROL sc:

```
var comm_array = sc.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of SensorControl objects for all of the flagged *SENSOR_CONTROLs in a model in PRIMER If the optional property argument is not given then an array of SensorControl objects is returned. If the property argument is given, then that property value for each *SENSOR_CONTROL is returned in the array instead of a SensorControl object

Arguments

- **Model** (Model)

Model to get *SENSOR_CONTROLs from

- **flag** (Flag)

Flag set on the *SENSOR_CONTROLs that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged *SENSOR_CONTROLs in the model

Returns

Array of SensorControl objects or properties

Return type

Array

Example

To make an array of SensorControl objects for all of the *SENSOR_CONTROLs in model m flagged with f:

```
var sc = SensorControl.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the *SENSOR_CONTROLs in model m flagged with f:

```
var a = SensorControl.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the SensorControl object for a *SENSOR_CONTROL ID.

Arguments

- **Model** (Model)

Model to find the *SENSOR_CONTROL in

- **number** (integer)

number of the *SENSOR_CONTROL you want the SensorControl object for

Returns

SensorControl object (or null if *SENSOR_CONTROL does not exist).

Return type

SensorControl

Example

To get the SensorControl object for *SENSOR_CONTROL 100 in model m

```
var sc = SensorControl.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a SensorControl property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the SensorControl.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

*SENSOR_CONTROL property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if SensorControl property sc.example is a parameter:

```
Options.property_parameter_names = true;
if (sc.GetParameter(sc.example) ) do_something...
Options.property_parameter_names = false;
```

To check if SensorControl property sc.example is a parameter by using the GetParameter method:

```
if (sc.ViewParameters().GetParameter(sc.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this *SENSOR_CONTROL. **Note that a carriage return is not added**. See also SensorControl.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for SensorControl sc:

```
var key = sc.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the *SENSOR_CONTROL. **Note that a carriage return is not added**. See also SensorControl.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for sensor control sc:

```
var cards = sc.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last *SENSOR_CONTROL in the model.

Arguments

- **Model** (Model)

Model to get last *SENSOR_CONTROL in

Returns

SensorControl object (or null if there are no *SENSOR_CONTROLs in the model).

Return type

SensorControl

Example

To get the last *SENSOR_CONTROL in model m:

```
var sc = SensorControl.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free *SENSOR_CONTROL label in the model. Also see SensorControl.FirstFreeLabel(), SensorControl.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free *SENSOR_CONTROL label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

SensorControl label.

Return type

Number

Example

To get the last free *SENSOR_CONTROL label in model m:

```
var label = SensorControl.LastFreeLabel(m);
```

Next()

Description

Returns the next *SENSOR_CONTROL in the model.

Arguments

No arguments

Returns

SensorControl object (or null if there are no more *SENSOR_CONTROLs in the model).

Return type

SensorControl

Example

To get the *SENSOR_CONTROL in model m after *SENSOR_CONTROL sc:

```
var sc = sc.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) *SENSOR_CONTROL label in the model. Also see SensorControl.FirstFreeLabel(), SensorControl.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free *SENSOR_CONTROL label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

SensorControl label.

Return type

Number

Example

To get the next free *SENSOR_CONTROL label in model m:

```
var label = SensorControl.NextFreeLabel(m);
```

Previous()

Description

Returns the previous *SENSOR_CONTROL in the model.

Arguments

No arguments

Returns

SensorControl object (or null if there are no more *SENSOR_CONTROLs in the model).

Return type

SensorControl

Example

To get the *SENSOR_CONTROL in model m before *SENSOR_CONTROL sc:

```
var sc = sc.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the *SENSOR_CONTROLs in the model.

Arguments

- **Model** (Model)

Model that all *SENSOR_CONTROLs will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the *SENSOR_CONTROLs in model m, from 1000000:

```
SensorControl.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged *SENSOR_CONTROLs in the model.

Arguments

- **Model** (Model)

Model that all the flagged *SENSOR_CONTROLs will be renumbered in

- **flag** (Flag)

Flag set on the *SENSOR_CONTROLs that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the *SENSOR_CONTROLs in model m flagged with f, from 1000000:

```
SensorControl.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select *SENSOR_CONTROLs using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting *SENSOR_CONTROLs

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only *SENSOR_CONTROLs from that model can be selected. If the argument is a Flag then only *SENSOR_CONTROLs that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any *SENSOR_CONTROLs can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of *SENSOR_CONTROLs selected or null if menu cancelled

Return type

Number

Example

To select *SENSOR_CONTROLs from model m, flagging those selected with flag f, giving the prompt 'Select *SENSOR_CONTROLs':

```
SensorControl.Select(f, 'Select *SENSOR_CONTROLs', m);
```

To select *SENSOR_CONTROLs, flagging those selected with flag f but limiting selection to *SENSOR_CONTROLs flagged with flag l, giving the prompt 'Select *SENSOR_CONTROLs':

```
SensorControl.Select(f, 'Select *SENSOR_CONTROLs', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the *SENSOR_CONTROL.

Arguments

- **flag** (Flag)

Flag to set on the *SENSOR_CONTROL

Returns

No return value

Example

To set flag f for *SENSOR_CONTROL sc:

```
sc.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the *SENSOR_CONTROL. The *SENSOR_CONTROL will be sketched until you either call SensorControl.Unsketch(), SensorControl.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the *SENSOR_CONTROL is sketched. If omitted redraw is true. If you want to sketch several *SENSOR_CONTROLs and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch *SENSOR_CONTROL sc:

```
sc.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged *SENSOR_CONTROLs in the model. The *SENSOR_CONTROLs will be sketched until you either call SensorControl.Unsketch(), SensorControl.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged *SENSOR_CONTROLs will be sketched in

- **flag** (Flag)

Flag set on the *SENSOR_CONTROLs that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the *SENSOR_CONTROLs are sketched. If omitted redraw is true. If you want to sketch flagged *SENSOR_CONTROLs several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all *SENSOR_CONTROLs flagged with flag in model m:

```
SensorControl.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of *SENSOR_CONTROLs in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing *SENSOR_CONTROLs should be counted. If false or omitted referenced but undefined *SENSOR_CONTROLs will also be included in the total.

Returns

number of *SENSOR_CONTROLs

Return type

Number

Example

To get the total number of *SENSOR_CONTROLs in model m:

```
var total = SensorControl.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the *SENSOR_CONTROLs in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all *SENSOR_CONTROLs will be unset in

- **flag** (Flag)

Flag to unset on the *SENSOR_CONTROLs

Returns

No return value

Example

To unset the flag f on all the *SENSOR_CONTROLs in model m:

```
SensorControl.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the *SENSOR_CONTROL.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the *SENSOR_CONTROL is unsketched. If omitted redraw is true. If you want to unsketch several *SENSOR_CONTROLs and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch *SENSOR_CONTROL sc:

```
sc.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all *SENSOR_CONTROLs.

Arguments

- **Model** (Model)

Model that all *SENSOR_CONTROLs will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the *SENSOR_CONTROLs are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all *SENSOR_CONTROLs in model m:

```
SensorControl.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged *SENSOR_CONTROLs in the model.

Arguments

- **Model** (Model)

Model that all *SENSOR_CONTROLs will be unsketched in

- **flag** (Flag)

Flag set on the *SENSOR_CONTROLs that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the *SENSOR_CONTROLs are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all *SENSOR_CONTROLs flagged with flag in model m:

```
SensorControl.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

SensorControl object.

Return type

SensorControl

Example

To check if SensorControl property sc.example is a parameter by using the SensorControl.GetParameter() method:

```
if (sc.ViewParameters().GetParameter(sc.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for *SENSOR_CONTROL with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The *SENSOR_CONTROL object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for *SENSOR_CONTROL sc:

```
sc.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for *SENSOR_CONTROL sc:

```
sc.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for *SENSOR_CONTROL. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for *SENSOR_CONTROL sc:

```
sc.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this *SENSOR_CONTROL.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for *SENSOR_CONTROL sc:

```
var xrefs = sc.Xrefs();
```

---

toString()

Description

Creates a string containing the sensor control data in keyword format. Note that this contains the keyword header and the keyword cards. See also SensorControl.Keyword() and SensorControl.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for sensor control sc in keyword format

```
var str = sc.toString();
```

SensorDefine class

The SensorDefine class gives you access to *SENSOR_DEFINE keyword in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

SensorDefine constants

Name	Description
SensorDefine.DEFINE_CALC_MATH	Sensor define is *SENSOR_DEFINE_CALC_MATH.
SensorDefine.DEFINE_ELEMENT	Sensor define is *SENSOR_DEFINE_ELEMENT.
SensorDefine.DEFINE_ELEMENT_SET	Sensor define is *SENSOR_DEFINE_ELEMENT_SET.

SensorDefine.DEFINE_FORCE	Sensor define is *SENSOR_DEFINE_FORCE.
SensorDefine.DEFINE_FUNCTION	Sensor define is *SENSOR_DEFINE_FUNCTION.
SensorDefine.DEFINE_MISC	Sensor define is *SENSOR_DEFINE_MISC.
SensorDefine.DEFINE_NODE	Sensor define is *SENSOR_DEFINE_NODE.
SensorDefine.DEFINE_NODE_SET	Sensor define is *SENSOR_DEFINE_NODE_SET.

SensorDefine properties

Name	Type	Description
calc	string	Mathematical calculation. Can be "ABSSUM", "MIN", "MAX", "MAXMAG", "MINMAG", "MULTIPLY", "SQRE", "SQRTSQRE", "SQRT", "SUMABS", "SUM" .
comp	string	Component type. Can be "XX", "YY", "ZZ", "XY", "YZ", "ZX", "HYDR", "MAXS", "PRIN1", "PRIN2", PRIN3", "VM", "F[ID]", "AXIAL", "SHEARS", "SHEART".
crd	integer	Optional coordinate system.
ctype	string	Sensor type or Output component type. Can be "STRAIN", "STRESS", "FORCE", "MOMENT", "DLEN" or "FAIL" for SensorDefine.DEFINE_ELEMENT or SensorDefine.DEFINE_ELEMENT_SET and "ACC", "VEL", "COORD" or "TEMP" for SensorDefine.DEFINE_NODE or SensorDefine.DEFINE_NODE_SET
elemid	integer	Element ID or element set ID when option_SET is active.
etype	string	Element type. Can be "BEAM", "SHELL", "SOLID", "DISC-ELE", "SEATBELT" or "TSHELL".
exists (read only)	logical	true if *SENSOR_DEFINE exists, false if referred to but not defined.
ftype	string	Force type. Can be "AIRBAG", "CONTACT", "CONTACT2D", "CPM", "JOINT", "JOINTSTIF", "PRESC-MOT", "RWALL", "SPC", "SPOTWELD", "X-SECTION".
func	integer	Function ID.
func_ sens1	integer	1st Sensor ID if positive or number of sensor ID if negative.
func_ sens10	integer	10th Sensor ID.
func_ sens11	integer	11th Sensor ID.
func_ sens12	integer	12th Sensor ID.
func_ sens13	integer	13th Sensor ID.
func_ sens14	integer	14th Sensor ID.
func_ sens15	integer	15th Sensor ID.
func_ sens16	integer	16th Sensor ID.
func_ sens2	integer	2nd Sensor ID.
func_ sens3	integer	3rd Sensor ID.

func_ sens4	integer	4th Sensor ID.
func_ sens5	integer	5th Sensor ID.
func_ sens6	integer	6th Sensor ID.
func_ sens7	integer	7th Sensor ID.
func_ sens8	integer	8th Sensor ID.
func_ sens9	integer	9th Sensor ID.
i0	string	I0. Can be "ADDMASS", "KINETIC", "INTERNAL", "ERODEKE", or "ERODEIE" when MTYPE = "MATSUM" or "SOLID", "SHELL", "TSHELL", "BEAM", or "DISC" when MTYPE = "NFAILE" or "TEMP", or "VOL" when MTYPE = "CVBAG", or "PRES", or "VOL" when MTYPE = "ICVOL", or "PULLRATE", "PULLOUT" or "FORCE" when MTYPE = "RETRACTOR" or "CRNT", "CRNTRTE", "FX", "FY" or "FZ" when MTYPE = "EMROGO" or "SLIP", "WRAP", "SKEW", "FC", "FN", "FSB1" or FSB2 when MTYPE = "SLIPRING" or "SLIP", "WRAP", "FN", "FBM1" or "FBM2" when MTYPE = "PULLEY".
i1	integer/string	I1. Applicable for "ANGLE", "BNDOUT", "CURVE", "CVBAG", "ICVOL", "MATSUM", "NFAILE", "EMROGO", "SLIPRING" or "PULLEY".
i2	integer/string	I2. Applicable only for when MTYPE = "ANGLE".
i3	integer/string	I3. Applicable only for MTYPE = "ANGLE".
i4	integer/string	I4. Applicable only for MTYPE = "ANGLE".
i5	string	I5.
include	integer	The Include file number that the *SENSOR_DEFINE is in.
label	integer	SensorDefine number. The sensid property is an alternative name for this.
layer	integer/string	Layer of integration. Can be "BOT", "TOP" or "i" to monitor the stress of the ith integration point when ctype = "STRESS".
model (read only)	integer	The Model number that the *SENSOR_DEFINE is in.
mtype	string	Entity to be traced. Can be "ANGLE", "BNDOUT", "CURVE", "CVBAG", "ICVOL",'MATSUM', "NFAILE", "RETRACTOR", "RIGIDBODY" .
node1	integer	Node or Node set ID based on option SET for an accelerometer sensor.
node2	integer	Node ID for an accelerometer sensor.
option	constant	SENSOR_DEFINE suffix. Can be SensorDefine.DEFINE_CALC_MATH, SensorDefine.DEFINE_ELEMENT, SensorDefine.DEFINE_ELEMENT_SET, SensorDefine.DEFINE_FORCE, SensorDefine.DEFINE_MISC, SensorDefine.DEFINE_ NODE, SensorDefine.DEFINE_NODE_SET or SensorDefine.DEFINE_FUNCTION.
pwr	real	Power (Optional parameters).
sens1	integer	1st Sensor ID.
sens2	integer	2nd Sensor ID.
sens3	integer	3rd Sensor ID.
sens4	integer	4th Sensor ID.
sens5	integer	5th Sensor ID.
sens6	integer	6th Sensor ID.
sensid	integer	SensorDefine number. The label property is an alternative name for this.

setopt	string	Option to process set of data when SET option is specified. Can be "AVG", "MAX", "MIN" or "SUM".
sf	real	Scale factor (Optional parameters).
typeid	integer	ID defined in the associated KEYWORD command.
vid	integer/string	Vector along which the forces is measured. Can be "X", "Y", "Z", "XL", "YL", "ZL", "XMOMENT", "YMOMENT", "ZMOMENT", "XLMOMENT", "YLMOMENT", "ZLMOMENT" or vector ID n in coordinate system CRD for SensorDefine.DEFINE_FORCE or ID of vector along which the nodal values for SensorDefine.DEFINE_NODE and SensorDefine.DEFINE_NODE_SET.

Detailed Description

The SensorDefine class allows you to create, modify, edit and manipulate *SENSOR_DEFINE. See the documentation below for more details.

Constructor

new SensorDefine(Option*[constant]*, Model*[Model]*, Define ID*[integer]*, Type or Entity 1*[string/label]*, Entity 2*[label]*)

Description

Create a new SensorDefine object.

Arguments

- **Option** (constant)

SENSOR_DEFINE suffix. Can be SensorDefine.DEFINE_CALC_MATH, SensorDefine.DEFINE_ELEMENT, SensorDefine.DEFINE_ELEMENT_SET, SensorDefine.DEFINE_FORCE, SensorDefine.DEFINE_MISC, SensorDefine.DEFINE_NODE, SensorDefine.DEFINE_NODE_SET or SensorDefine.DEFINE_FUNCTION.

- **Model** (Model)

Model that *SENSOR_DEFINE will be created in

- **Define ID** (integer)

SensorDefine id.

- **Type or Entity 1** (string/label)

For SensorDefine.DEFINE_NODE, SensorDefine.DEFINE_NODE_SET option it is Node ID or NODE set ID respectively, For SensorDefine.DEFINE_FUNCTION option it is DEFINE_FUNCTION ID, For SensorDefine.DEFINE_CALC_MATH option it is Calc string, For SensorDefine.DEFINE_ELEMENT and SensorDefine.DEFINE_ELEMENT_SET option it is Etype string, For SensorDefine.DEFINE_FORCE option it is Ftype string, For SensorDefine.DEFINE_MISC option it is Mtype string.

- **Entity 2** (label)

Applicable only for SensorDefine.DEFINE_NODE, SensorDefine.DEFINE_NODE_SET, SensorDefine.DEFINE_CALC_MATH, SensorDefine.DEFINE_ELEMENT, SensorDefine.DEFINE_ELEMENT_SET or SensorDefine.DEFINE_FORCE. It is NODE or NODE set ID for SensorDefine.DEFINE_NODE or SensorDefine.DEFINE_NODE_SET respectively, Sensor Define ID for SensorDefine.DEFINE_CALC_MATH, Element ID or Element set ID for SensorDefine.DEFINE_ELEMENT or SensorDefine.DEFINE_ELEMENT_SET respectively or Type ID for SensorDefine.DEFINE_FORCE.

Returns

SensorDefine object

Return type

SensorDefine

Example

To create a new *SENSOR_DEFINE_CALC-MATH in model m with label 100 with CALC option as MAX and SENS1 as -2:

```
var sd1 = new SensorDefine(SensorDefine.DEFINE_CALC_MATH, m, 100, "MAX", -2);
```

To create a new *SENSOR_DEFINE_MISC in model m with label 10 with MTYPE option as ANGLE:

```
var sd2 = new SensorDefine(SensorDefine.DEFINE_MISC, m, 10, "ANGLE");
```

To create a new *SENSOR_DEFINE_NODE in model m with label 11 with NODE1 and NODE2 as 5 and 6:

```
var sd3 = new SensorDefine(SensorDefine.DEFINE_NODE, m, 11, 5, 6);
```

To create a new *SENSOR_DEFINE_FUNCTION in model m with label 12 and FUNCTION ID as 6:

```
var sd4 = new SensorDefine(SensorDefine.DEFINE_FUNCTION, m, 12, 6);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a *SENSOR_DEFINE.

Arguments

- **Comment** (Comment)

Comment that will be attached to the *SENSOR_DEFINE

Returns

No return value

Example

To associate comment c to the *SENSOR_DEFINE sd:

```
sd.AssociateComment(c);
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse *SENSOR_DEFINE sd:

```
sd.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the *SENSOR_DEFINE.

Arguments

- **flag** (Flag)

Flag to clear on the *SENSOR_DEFINE

Returns

No return value

Example

To clear flag f for *SENSOR_DEFINE sd:

```
sd.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the *SENSOR_DEFINE. The target include of the copied *SENSOR_DEFINE can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

SensorDefine object

Return type

SensorDefine

Example

To copy *SENSOR_DEFINE sd into *SENSOR_DEFINE z:

```
var z = sd.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a *SENSOR_DEFINE

Arguments

- **Model** (Model)

Model that the *SENSOR_DEFINE will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

SensorDefine object (or null if not made).

Return type

SensorDefine

Example

To start creating a *SENSOR_DEFINE in model m:

```
var sd = SensorDefine.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a *SENSOR_DEFINE.

Arguments

- **Comment** (Comment)

Comment that will be detached from the *SENSOR_DEFINE

Returns

No return value

Example

To detach comment c from the *SENSOR_DEFINE sd:

```
sd.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit *SENSOR_DEFINE sd:

```
sd.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for *SENSOR_DEFINE with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The *SENSOR_DEFINE object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for *SENSOR_DEFINE sd:

```
sd.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for *SENSOR_DEFINE sd:

```
sd.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for *SENSOR_DEFINE. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for *SENSOR_DEFINE sd:

```
sd.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first *SENSOR_DEFINE in the model.

Arguments

- **Model** (Model)

Model to get first *SENSOR_DEFINE in

Returns

SensorDefine object (or null if there are no *SENSOR_DEFINEs in the model).

Return type

SensorDefine

Example

To get the first *SENSOR_DEFINE in model m:

```
var sd = SensorDefine.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free *SENSOR_DEFINE label in the model. Also see SensorDefine.LastFreeLabel(), SensorDefine.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free *SENSOR_DEFINE label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

SensorDefine label.

Return type

Number

Example

To get the first free *SENSOR_DEFINE label in model m:

```
var label = SensorDefine.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the *SENSOR_DEFINEs in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all *SENSOR_DEFINEs will be flagged in

- **flag** (Flag)

Flag to set on the *SENSOR_DEFINEs

Returns

No return value

Example

To flag all of the *SENSOR_DEFINEs with flag f in model m:

```
SensorDefine.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the *SENSOR_DEFINE is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the *SENSOR_DEFINE

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if *SENSOR_DEFINE sd has flag f set on it:

```
if (sd.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each *SENSOR_DEFINE in the model.
Note that ForEach has been designed to make looping over *SENSOR_DEFINEs as fast as possible and so has some limitations.
Firstly, a single temporary SensorDefine object is created and on each function call it is updated with the current *SENSOR_DEFINE data. This means that you should not try to store the SensorDefine object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new *SENSOR_DEFINEs inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all *SENSOR_DEFINEs are in

- **func** (function)

Function to call for each *SENSOR_DEFINE

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the *SENSOR_DEFINEs in model m:

```
SensorDefine.ForEach(m, test);
function test(sd)
{
// sd is SensorDefine object
}
```

To call function test for all of the *SENSOR_DEFINEs in model m with optional object:

```
var data = { x:0, y:0 };
SensorDefine.ForEach(m, test, data);
function test(sd, extra)
{
// sd is SensorDefine object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of SensorDefine objects or properties for all of the *SENSOR_DEFINEs in a model in PRIMER. If the optional property argument is not given then an array of SensorDefine objects is returned. If the property argument is given, that property value for each *SENSOR_DEFINE is returned in the array instead of a SensorDefine object

Arguments

- **Model** (Model)

Model to get *SENSOR_DEFINEs from

- **property (optional)** (string)

Name for property to get for all *SENSOR_DEFINEs in the model

Returns

Array of SensorDefine objects or properties

Return type

Array

Example

To make an array of SensorDefine objects for all of the *SENSOR_DEFINEs in model m:

```
var a = SensorDefine.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each *SENSOR_DEFINE in model m:

```
var a = SensorDefine.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a *SENSOR_DEFINE.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the *SENSOR_DEFINE sd:

```
var comm_array = sd.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of SensorDefine objects for all of the flagged *SENSOR_DEFINEs in a model in PRIMER If the optional property argument is not given then an array of SensorDefine objects is returned. If the property argument is given, then that property value for each *SENSOR_DEFINE is returned in the array instead of a SensorDefine object

Arguments

- **Model** (Model)

Model to get *SENSOR_DEFINEs from

- **flag** (Flag)

Flag set on the *SENSOR_DEFINEs that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged *SENSOR_DEFINEs in the model

Returns

Array of SensorDefine objects or properties

Return type

Array

Example

To make an array of SensorDefine objects for all of the *SENSOR_DEFINEs in model m flagged with f:

```
var sd = SensorDefine.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the *SENSOR_DEFINEs in model m flagged with f:

```
var a = SensorDefine.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the SensorDefine object for a *SENSOR_DEFINE ID.

Arguments

- **Model** (Model)

Model to find the *SENSOR_DEFINE in

- **number** (integer)

number of the *SENSOR_DEFINE you want the SensorDefine object for

Returns

SensorDefine object (or null if *SENSOR_DEFINE does not exist).

Return type

SensorDefine

Example

To get the SensorDefine object for *SENSOR_DEFINE 100 in model m

```
var sd = SensorDefine.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a SensorDefine property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the SensorDefine.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

*SENSOR_DEFINE property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if SensorDefine property sd.example is a parameter:

```
Options.property_parameter_names = true;
if (sd.GetParameter(sd.example) ) do_something...
Options.property_parameter_names = false;
```

To check if SensorDefine property sd.example is a parameter by using the GetParameter method:

```
if (sd.ViewParameters().GetParameter(sd.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this *SENSOR_DEFINE. **Note that a carriage return is not added**. See also SensorDefine.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for SensorDefine sd:

```
var key = sd.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the *SENSOR_DEFINE. **Note that a carriage return is not added**. See also SensorDefine.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for sensor define sd:

```
var cards = sd.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last *SENSOR_DEFINE in the model.

Arguments

- **Model** (Model)

Model to get last *SENSOR_DEFINE in

Returns

SensorDefine object (or null if there are no *SENSOR_DEFINEs in the model).

Return type

SensorDefine

Example

To get the last *SENSOR_DEFINE in model m:

```
var sd = SensorDefine.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free *SENSOR_DEFINE label in the model. Also see SensorDefine.FirstFreeLabel(), SensorDefine.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free *SENSOR_DEFINE label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

SensorDefine label.

Return type

Number

Example

To get the last free *SENSOR_DEFINE label in model m:

```
var label = SensorDefine.LastFreeLabel(m);
```

Next()

Description

Returns the next *SENSOR_DEFINE in the model.

Arguments

No arguments

Returns

SensorDefine object (or null if there are no more *SENSOR_DEFINEs in the model).

Return type

SensorDefine

Example

To get the *SENSOR_DEFINE in model m after *SENSOR_DEFINE sd:

```
var sd = sd.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) *SENSOR_DEFINE label in the model. Also see SensorDefine.FirstFreeLabel(), SensorDefine.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free *SENSOR_DEFINE label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

SensorDefine label.

Return type

Number

Example

To get the next free *SENSOR_DEFINE label in model m:

```
var label = SensorDefine.NextFreeLabel(m);
```

Previous()

Description

Returns the previous *SENSOR_DEFINE in the model.

Arguments

No arguments

Returns

SensorDefine object (or null if there are no more *SENSOR_DEFINEs in the model).

Return type

SensorDefine

Example

To get the *SENSOR_DEFINE in model m before *SENSOR_DEFINE sd:

```
var sd = sd.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the *SENSOR_DEFINEs in the model.

Arguments

- **Model** (Model)

Model that all *SENSOR_DEFINEs will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the *SENSOR_DEFINEs in model m, from 1000000:

```
SensorDefine.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged *SENSOR_DEFINEs in the model.

Arguments

- **Model** (Model)

Model that all the flagged *SENSOR_DEFINEs will be renumbered in

- **flag** (Flag)

Flag set on the *SENSOR_DEFINEs that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the *SENSOR_DEFINEs in model m flagged with f, from 1000000:

```
SensorDefine.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select *SENSOR_DEFINEs using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting *SENSOR_DEFINEs

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only *SENSOR_DEFINEs from that model can be selected. If the argument is a Flag then only *SENSOR_DEFINEs that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any *SENSOR_DEFINEs can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of *SENSOR_DEFINEs selected or null if menu cancelled

Return type

Number

Example

To select *SENSOR_DEFINEs from model m, flagging those selected with flag f, giving the prompt 'Select *SENSOR_DEFINEs':

```
SensorDefine.Select(f, 'Select *SENSOR_DEFINEs', m);
```

To select *SENSOR_DEFINEs, flagging those selected with flag f but limiting selection to *SENSOR_DEFINEs flagged with flag l, giving the prompt 'Select *SENSOR_DEFINEs':

```
SensorDefine.Select(f, 'Select *SENSOR_DEFINEs', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the *SENSOR_DEFINE.

Arguments

- **flag** (Flag)

Flag to set on the *SENSOR_DEFINE

Returns

No return value

Example

To set flag f for *SENSOR_DEFINE sd:

```
sd.SetFlag(f);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of *SENSOR_DEFINEs in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing *SENSOR_DEFINEs should be counted. If false or omitted referenced but undefined *SENSOR_DEFINEs will also be included in the total.

Returns

number of *SENSOR_DEFINEs

Return type

Number

Example

To get the total number of *SENSOR_DEFINEs in model m:

```
var total = SensorDefine.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the *SENSOR_DEFINEs in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all *SENSOR_DEFINEs will be unset in

- **flag** (Flag)

Flag to unset on the *SENSOR_DEFINEs

Returns

No return value

Example

To unset the flag f on all the *SENSOR_DEFINEs in model m:

```
SensorDefine.UnflagAll(m, f);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

SensorDefine object.

Return type

SensorDefine

Example

To check if SensorDefine property sd.example is a parameter by using the SensorDefine.GetParameter() method:

```
if (sd.ViewParameters().GetParameter(sd.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for *SENSOR_DEFINE with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The *SENSOR_DEFINE object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for *SENSOR_DEFINE sd:

```
sd.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for *SENSOR_DEFINE sd:

```
sd.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for *SENSOR_DEFINE. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for *SENSOR_DEFINE sd:

```
sd.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this *SENSOR_DEFINE.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for *SENSOR_DEFINE sd:

```
var xrefs = sd.Xrefs();
```

---

toString()

Description

Creates a string containing the sensor define data in keyword format. Note that this contains the keyword header and the keyword cards. See also SensorDefine.Keyword() and SensorDefine.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for sensor define sd in keyword format

```
var str = sd.toString();
```

SensorSwitch class

The SensorSwitch class gives you access to *SENSOR_SWITCH keyword in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- GetRow(row*[integer]*)
- GetSwitch(row*[integer]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- RemoveRow(row*[integer]*)
- RemoveSwitch(row*[integer]*)
- SetFlag(flag*[Flag]*)
- SetRow(row*[integer]*, data*[Array of data]*)
- SetSwitch(index*[integer]*, data*[object]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

SensorSwitch constants

Name	Description
SensorSwitch.SWITCH	Sensor switch is *SENSOR_SWITCH.
SensorSwitch.SWITCH_CALC_LOGIC	Sensor switch is *SENSOR_SWITCH_CALC-LOGIC.
SensorSwitch.SWITCH_SHELL_TO_VENT	Sensor switch is *SENSOR_SWITCH_SHELL_TO_VENT.

SensorSwitch properties

Name	Type	Description
abid	integer	Airbag ID.
amax	real	Maximum allowable area for failed vent surface area (VA).
c23	integer/real	Vent coefficient if positive or user defined load curve ID if negative.
exists (read only)	logical	true if *SENSOR_SWITCH exists, false if referred to but not defined.
filtrid	integer	Filter ID.
id	integer	Part set ID or Part ID.
id_flag	logical	Turns _TITLE/_ID ON or OFF. Used only for SensorSwitch.SWITCH_SHELL_TO_VENT.
include	integer	The Include file number that the *SENSOR_SWITCH is in.
itype	integer	0 for Part, 1 for Part Set.
label	integer	SensorSwitch number. The switid property is an alternative name for this.
logic	string	Logic operator.
model (read only)	integer	The Model number that the *SENSOR_SWITCH is in.
nrow	integer	Number of Shell Fail Time Cards.
nswit (read only)	integer	Number of sensor switch IDs defined. IDs can be positive for "AND", negative ID for "OR". Applicable to SensorSwitch.SWITCH_CALC_LOGIC.
option	constant	SENSOR_SWITCH suffix. Can be SensorSwitch.SWITCH, SensorSwitch.SWITCH_CALC_LOGIC or SensorSwitch.SWITCH_SHELL_TO_VENT.
sensid	integer	ID of the sensor whose value will be compared.
switid	integer	SensorSwitch number. The label property is an alternative name for this.
timwin	real	Trigger status change when the value given by the sensor is less/greater (depending on logic) than value for duration defined by timwin.
title	string	SensorSwitch title. Used only for SensorSwitch.SWITCH_SHELL_TO_VENT.
value	real	Critical value.

Detailed Description

The SensorSwitch class allows you to create, modify, edit and manipulate *SENSOR_SWITCH. See the documentation below for more details.

Constructor

new SensorSwitch(Option*[constant]*, Model*[Model]*, Switch ID*[integer]*)

Description

Create a new SensorSwitch object.

Arguments

- **Option** (constant)

SENSOR_SWITCH suffix. Can be SensorSwitch.SWITCH, SensorSwitch.SWITCH_CALC_LOGIC or SensorSwitch.SWITCH_SHELL_TO_VENT.

- **Model** (Model)

Model that *SENSOR_SWITCH will be created in

- **Switch ID** (integer)

SensorSwitch id. This is required for the SensorSwitch.SWITCH and SensorSwitch.SWITCH_CALC_LOGIC options and ignored for SensorSwitch.SWITCH_SHELL_TO_VENT.

Returns

SensorSwitch object

Return type

SensorSwitch

Example

To create a new *SENSOR_SWITCH in model m with label 100:

```
var sc = new SensorSwitch(SensorSwitch.SWITCH, m, 100);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a *SENSOR_SWITCH.

Arguments

- **Comment** (Comment)

Comment that will be attached to the *SENSOR_SWITCH

Returns

No return value

Example

To associate comment c to the *SENSOR_SWITCH ss:

```
ss.AssociateComment(c);
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse *SENSOR_SWITCH ss:

```
ss.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the *SENSOR_SWITCH.

Arguments

- **flag** (Flag)

Flag to clear on the *SENSOR_SWITCH

Returns

No return value

Example

To clear flag f for *SENSOR_SWITCH ss:

```
ss.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the *SENSOR_SWITCH. The target include of the copied *SENSOR_SWITCH can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

SensorSwitch object

Return type

SensorSwitch

Example

To copy *SENSOR_SWITCH ss into *SENSOR_SWITCH z:

```
var z = ss.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a *SENSOR_SWITCH

Arguments

- **Model** (Model)

Model that the *SENSOR_SWITCH will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

SensorSwitch object (or null if not made).

Return type

SensorSwitch

Example

To start creating a *SENSOR_SWITCH in model m:

```
var ss = SensorSwitch.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a *SENSOR_SWITCH.

Arguments

- **Comment** (Comment)

Comment that will be detached from the *SENSOR_SWITCH

Returns

No return value

Example

To detach comment c from the *SENSOR_SWITCH ss:

```
ss.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit *SENSOR_SWITCH ss:

```
ss.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for *SENSOR_SWITCH with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The *SENSOR_SWITCH object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for *SENSOR_SWITCH ss:

```
ss.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for *SENSOR_SWITCH ss:

```
ss.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for *SENSOR_SWITCH. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for *SENSOR_SWITCH ss:

```
ss.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first *SENSOR_SWITCH in the model.

Arguments

- **Model** (Model)

Model to get first *SENSOR_SWITCH in

Returns

SensorSwitch object (or null if there are no *SENSOR_SWITCHs in the model).

Return type

SensorSwitch

Example

To get the first *SENSOR_SWITCH in model m:

```
var ss = SensorSwitch.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free *SENSOR_SWITCH label in the model. Also see SensorSwitch.LastFreeLabel(), SensorSwitch.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free *SENSOR_SWITCH label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

SensorSwitch label.

Return type

Number

Example

To get the first free *SENSOR_SWITCH label in model m:

```
var label = SensorSwitch.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the *SENSOR_SWITCHs in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all *SENSOR_SWITCHs will be flagged in

- **flag** (Flag)

Flag to set on the *SENSOR_SWITCHs

Returns

No return value

Example

To flag all of the *SENSOR_SWITCHs with flag f in model m:

```
SensorSwitch.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the *SENSOR_SWITCH is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the *SENSOR_SWITCH

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if *SENSOR_SWITCH ss has flag f set on it:

```
if (ss.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each *SENSOR_SWITCH in the model.
Note that ForEach has been designed to make looping over *SENSOR_SWITCHs as fast as possible and so has some limitations.
Firstly, a single temporary SensorSwitch object is created and on each function call it is updated with the current *SENSOR_SWITCH data. This means that you should not try to store the SensorSwitch object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new *SENSOR_SWITCHs inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all *SENSOR_SWITCHs are in

- **func** (function)

Function to call for each *SENSOR_SWITCH

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the *SENSOR_SWITCHs in model m:

```
SensorSwitch.ForEach(m, test);
function test(ss)
{
// ss is SensorSwitch object
}
```

To call function test for all of the *SENSOR_SWITCHs in model m with optional object:

```
var data = { x:0, y:0 };
SensorSwitch.ForEach(m, test, data);
function test(ss, extra)
{
// ss is SensorSwitch object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of SensorSwitch objects or properties for all of the *SENSOR_SWITCHs in a model in PRIMER. If the optional property argument is not given then an array of SensorSwitch objects is returned. If the property argument is given, that property value for each *SENSOR_SWITCH is returned in the array instead of a SensorSwitch object

Arguments

- **Model** (Model)

Model to get *SENSOR_SWITCHs from

- **property (optional)** (string)

Name for property to get for all *SENSOR_SWITCHs in the model

Returns

Array of SensorSwitch objects or properties

Return type

Array

Example

To make an array of SensorSwitch objects for all of the *SENSOR_SWITCHs in model m:

```
var a = SensorSwitch.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each *SENSOR_SWITCH in model m:

```
var a = SensorSwitch.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a *SENSOR_SWITCH.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the *SENSOR_SWITCH ss:

```
var comm_array = ss.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of SensorSwitch objects for all of the flagged *SENSOR_SWITCHs in a model in PRIMER If the optional property argument is not given then an array of SensorSwitch objects is returned. If the property argument is given, then that property value for each *SENSOR_SWITCH is returned in the array instead of a SensorSwitch object

Arguments

- **Model** (Model)

Model to get *SENSOR_SWITCHs from

- **flag** (Flag)

Flag set on the *SENSOR_SWITCHs that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged *SENSOR_SWITCHs in the model

Returns

Array of SensorSwitch objects or properties

Return type

Array

Example

To make an array of SensorSwitch objects for all of the *SENSOR_SWITCHs in model m flagged with f:

```
var ss = SensorSwitch.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the *SENSOR_SWITCHs in model m flagged with f:

```
var a = SensorSwitch.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the SensorSwitch object for a *SENSOR_SWITCH ID.

Arguments

- **Model** (Model)

Model to find the *SENSOR_SWITCH in

- **number** (integer)

number of the *SENSOR_SWITCH you want the SensorSwitch object for

Returns

SensorSwitch object (or null if *SENSOR_SWITCH does not exist).

Return type

SensorSwitch

Example

To get the SensorSwitch object for *SENSOR_SWITCH 100 in model m

```
var ss = SensorSwitch.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a SensorSwitch property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the SensorSwitch.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

*SENSOR_SWITCH property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if SensorSwitch property ss.example is a parameter:

```
Options.property_parameter_names = true;
if (ss.GetParameter(ss.example) ) do_something...
Options.property_parameter_names = false;
```

To check if SensorSwitch property ss.example is a parameter by using the GetParameter method:

```
if (ss.ViewParameters().GetParameter(ss.example) ) do_something...
```

GetRow(row*[integer]*)

Description

Returns the data for a row in the SENSOR_SWITCH_SHELL_TO_VENT.

Arguments

- **row** (integer)

The row you want the data for. **Note row indices start at 0**.

Returns

An array of numbers containing the row variables SSID, FTIME and C23V.

Return type

Number

Example

To get the data for the 2nd row in sensor switch ss:

```
var data = ss.GetRow(1);
```

GetSwitch(row*[integer]*)

Description

Returns switch ID information for *SENSOR_SWITCH_CALC-LOGIC.

Arguments

- **row** (integer)

The row you want the data for. **Note row indices start at 0**.

Returns

Object containing sensor switch ID information.

Return type

Object

Example

To get the data for the 2nd switch in sensor switch ss:

```
var data = ss.GetSwitch(1);
Message("Switch 2: " + data.swit);
```

Keyword()

Description

Returns the keyword for this *SENSOR_SWITCH. **Note that a carriage return is not added**. See also SensorSwitch.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for SensorSwitch ss:

```
var key = ss.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the *SENSOR_SWITCH. **Note that a carriage return is not added**. See also SensorSwitch.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for sensor switch ss:

```
var cards = ss.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last *SENSOR_SWITCH in the model.

Arguments

- **Model** (Model)

Model to get last *SENSOR_SWITCH in

Returns

SensorSwitch object (or null if there are no *SENSOR_SWITCHs in the model).

Return type

SensorSwitch

Example

To get the last *SENSOR_SWITCH in model m:

```
var ss = SensorSwitch.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free *SENSOR_SWITCH label in the model. Also see SensorSwitch.FirstFreeLabel(), SensorSwitch.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free *SENSOR_SWITCH label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

SensorSwitch label.

Return type

Number

Example

To get the last free *SENSOR_SWITCH label in model m:

```
var label = SensorSwitch.LastFreeLabel(m);
```

Next()

Description

Returns the next *SENSOR_SWITCH in the model.

Arguments

No arguments

Returns

SensorSwitch object (or null if there are no more *SENSOR_SWITCHs in the model).

Return type

SensorSwitch

Example

To get the *SENSOR_SWITCH in model m after *SENSOR_SWITCH ss:

```
var ss = ss.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) *SENSOR_SWITCH label in the model. Also see SensorSwitch.FirstFreeLabel(), SensorSwitch.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free *SENSOR_SWITCH label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

SensorSwitch label.

Return type

Number

Example

To get the next free *SENSOR_SWITCH label in model m:

```
var label = SensorSwitch.NextFreeLabel(m);
```

Previous()

Description

Returns the previous *SENSOR_SWITCH in the model.

Arguments

No arguments

Returns

SensorSwitch object (or null if there are no more *SENSOR_SWITCHs in the model).

Return type

SensorSwitch

Example

To get the *SENSOR_SWITCH in model m before *SENSOR_SWITCH ss:

```
var ss = ss.Previous();
```

RemoveRow(row*[integer]*)

Description

Removes the data for a row in *SENSOR_SWITCH_SHELL_TO_VENT.

Arguments

- **row** (integer)

The row you want to remove the data for. **Note that row indices start at 0**.

Returns

No return value.

Example

To remove the second row of data for sensor switch ss:

```
ss.RemoveRow(1);
```

RemoveSwitch(row*[integer]*)

Description

Removes sensor switch ID from *SENSOR_SWITCH_CALC-LOGIC.

Arguments

- **row** (integer)

The sensor switch ID that you want to remove. **Note that row indices start at 0**.

Returns

No return value.

Example

To remove the second sensor switch ID for sensor switch ss:

```
ss.RemoveSwitch(1);
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the *SENSOR_SWITCHs in the model.

Arguments

- **Model** (Model)

Model that all *SENSOR_SWITCHs will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the *SENSOR_SWITCHs in model m, from 1000000:

```
SensorSwitch.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged *SENSOR_SWITCHs in the model.

Arguments

- **Model** (Model)

Model that all the flagged *SENSOR_SWITCHs will be renumbered in

- **flag** (Flag)

Flag set on the *SENSOR_SWITCHs that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the *SENSOR_SWITCHs in model m flagged with f, from 1000000:

```
SensorSwitch.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select *SENSOR_SWITCHs using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting *SENSOR_SWITCHs

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only *SENSOR_SWITCHs from that model can be selected. If the argument is a Flag then only *SENSOR_SWITCHs that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any *SENSOR_SWITCHs can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of *SENSOR_SWITCHs selected or null if menu cancelled

Return type

Number

Example

To select *SENSOR_SWITCHs from model m, flagging those selected with flag f, giving the prompt 'Select *SENSOR_SWITCHs':

```
SensorSwitch.Select(f, 'Select *SENSOR_SWITCHs', m);
```

To select *SENSOR_SWITCHs, flagging those selected with flag f but limiting selection to *SENSOR_SWITCHs flagged with flag l, giving the prompt 'Select *SENSOR_SWITCHs':

```
SensorSwitch.Select(f, 'Select *SENSOR_SWITCHs', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the *SENSOR_SWITCH.

Arguments

- **flag** (Flag)

Flag to set on the *SENSOR_SWITCH

Returns

No return value

Example

To set flag f for *SENSOR_SWITCH ss:

```
ss.SetFlag(f);
```

SetRow(row*[integer]*, data*[Array of data]*)

Description

Sets the data for a row in *SENSOR_SWITCH_SHELL_TO_VENT.

Arguments

- **row** (integer)

The row you want to set the data for. **Note that row indices start at 0**.

- **data** (Array of data)

An array containing the row variables SSID, FTIME and C23V.

Returns

No return value.

Example

To set the second row of data for sensor switch ss to be shell set list 11, time 12.0 and vent coefficient 0.7:

```
var array = [11, 12.0, 0.7];
ss.SetRow(1, array);
```

To append a new row of data (using the same array of values):

```
ss.SetRow(ss.nrow, array);
```

SetSwitch(index*[integer]*, data*[object]*)

Description

Specifies a sensor switch ID for a *SENSOR_SWITCH_CALC-LOGIC.

Arguments

- **index** (integer)

The index of the *SENSOR_SWITCH_CALC-LOGIC data to set. **Note that indices start at 0, not 1**.
0 <= index <= nswit

- **data** (object)

Object containing sensor swith ID data.

Object has the following properties:

Name	Type	Description
swit	integer	Positive or negative sensor switch id.

Returns

No return value.

Example

To set the value of -10 for sensor switch 5 (indices start with 0) for *SENSOR_SWITCH_CALC-LOGIC s:

```
var data = { swit: -10 };
s.SetSwitch(4, data);
```

To append a new line of data (using the same example values):

```
var data2 = {swit: -10};
s.SetSwitch(b.lines, data2);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of *SENSOR_SWITCHs in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing *SENSOR_SWITCHs should be counted. If false or omitted referenced but undefined *SENSOR_SWITCHs will also be included in the total.

Returns

number of *SENSOR_SWITCHs

Return type

Number

Example

To get the total number of *SENSOR_SWITCHs in model m:

```
var total = SensorSwitch.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the *SENSOR_SWITCHs in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all *SENSOR_SWITCHs will be unset in

- **flag** (Flag)

Flag to unset on the *SENSOR_SWITCHs

Returns

No return value

Example

To unset the flag f on all the *SENSOR_SWITCHs in model m:

```
SensorSwitch.UnflagAll(m, f);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

SensorSwitch object.

Return type

SensorSwitch

Example

To check if SensorSwitch property ss.example is a parameter by using the SensorSwitch.GetParameter() method:

```
if (ss.ViewParameters().GetParameter(ss.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for *SENSOR_SWITCH with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The *SENSOR_SWITCH object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for *SENSOR_SWITCH ss:

```
ss.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for *SENSOR_SWITCH ss:

```
ss.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for *SENSOR_SWITCH. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for *SENSOR_SWITCH ss:

```
ss.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this *SENSOR_SWITCH.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for *SENSOR_SWITCH ss:

```
var xrefs = ss.Xrefs();
```

---

toString()

Description

Creates a string containing the sensor switch data in keyword format. Note that this contains the keyword header and the keyword cards. See also SensorSwitch.Keyword() and SensorSwitch.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for sensor switch ss in keyword format

```
var str = ss.toString();
```

Set (SetK) class

The Set class gives you access to sets in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, type (optional)*[constant]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, type (optional)*[constant]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, type*[constant]*, modal (optional)*[boolean]*)
- First(Model*[Model]*, type*[constant]*)
- FirstFreeLabel(Model*[Model]*, type*[constant]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*, type (optional)*[constant]*)
- GetAll(Model*[Model]*, type*[constant]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, type*[constant]*)
- GetFromID(Model*[Model]*, set number*[integer]*, type*[constant]*)
- Last(Model*[Model]*, type*[constant]*)
- LastFreeLabel(Model*[Model]*, type*[constant]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, type*[constant]*, layer (optional)*[Include number]*)
- Pick(type*[constant]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*, type (optional)*[constant]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*, type (optional)*[constant]*)
- Select(type*[constant]*, flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, type (optional)*[constant]*, redraw (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, type (optional)*[constant]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, type (optional)*[constant]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*, type (optional)*[constant]*)
- UnsketchAll(Model*[Model]*, type (optional)*[constant]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, type (optional)*[constant]*, redraw (optional)*[boolean]*)

Member functions

- Add(id1*[integer]*, id2 (optional)*[integer]*, id3 (optional)*[integer]*, id4 (optional)*[integer]*)
- AddCollectChild(set*[Set]*)
- AddFlagged(flag*[Flag]*)
- AllItems()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Contains(id*[integer]*)
- Copy(range (optional)*[boolean]*)
- Edit(modal (optional)*[boolean]*)
- Empty()
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetCollectChild(number*[Integer]*)
- GetGeneralData(index*[Integer]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- RebuildCache()
- Remove(id*[integer]*)
- RemoveFlagged(flag*[Flag]*)
- RemoveGeneralData(index*[Integer]*)
- SetFlag(flag*[Flag]*)

- SetGeneralData(index*[Integer]*, data*[Array of data]*)
- Sketch(redraw (optional)*[boolean]*)
- Spool()
- StartSpool(raw (optional)*[boolean]*)
- Unsketch(redraw (optional)*[boolean]*)
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

Set constants

Name	Description
Set.ADD	Set type is *SET_XYZ_ADD.
Set.ALL_TYPES	All set types - used in blanking etc.
Set.BEAM	Set beam type
Set.BOX	Set box type
Set.DISCRETE	Set discrete type
Set.GENERAL	Set type is *SET_XYZ_GENERAL.
Set.GENERATE	Set type is *SET_XYZ_GENERATE.
Set.IGA_EDGE	Set IGA edge uvw/xyz type
Set.IGA_FACE	Set IGA face uvw/xyz type
Set.IGA_POINT_UVW	Set IGA point uvw type
Set.IGA_UVW	Set UVW option for *SET_IGA_EDGE and *SET_IGA_FACE.
Set.IGA_XYZ	Set XYZ option for *SET_IGA_EDGE and *SET_IGA_FACE.
Set.INTERSECT	Set type is *SET_XYZ_INTERSECT.
Set.MM_GROUP	Set multi-material group type
Set.MODE	Set mode type
Set.NODE	Set node type
Set.PART	Set part type
Set.PART_TREE	Set part tree type
Set.PERI_LAMINATE	Set Peri Laminate type
Set.SEGMENT	Set segment type
Set.SEGMENT_2D	Set segment 2d type
Set.SHELL	Set shell type
Set.SOLID	Set solid type
Set.TSHELL	Set thick shell type

Set properties

Name	Type	Description
add	logical	If _ADD option is active.
advanced (read only)	logical	If _ADD_ADVANCED option is active.

collect	logical	If _COLLECT option is active. To manage _COLLECT sets PRIMER creates a 'parent' set that can be used to sketch/view all of the items from the _COLLECT sets with the same label. PRIMER then manages each _COLLECT set with the same label as a 'child' of this 'parent' set. Also see collect_children and GetCollectChild. If the collect property is unset for a child collect set then a new label will be assigned for the child set. If the collect property is unset for a parent collect set then all of the child sets will be reassigned new labels.
collect_ children (read only)	integer	The number of child _COLLECT sets if _COLLECT option is active.
colour	Colour	The colour of the set
column (read only)	logical	If _COLUMN option is active.
da1	real	The first default attribute for the set (only valid for Set.IGA_EDGE, Set.IGA_FACE, Set.IGA_POINT_UVW, Set.NODE, Set.PART, Set.SEGMENT, Set.SEGMENT_2D and Set.SHELL)
da2	real	The second default attribute for the set (only valid for Set.IGA_EDGE, Set.IGA_FACE, Set.IGA_POINT_UVW, Set.NODE, Set.PART, Set.SEGMENT, Set.SEGMENT_2D and Set.SHELL)
da3	real	The third default attribute for the set (only valid for Set.IGA_EDGE, Set.IGA_FACE, Set.IGA_POINT_UVW, Set.NODE, Set.PART, Set.SEGMENT, Set.SEGMENT_2D and Set.SHELL)
da4	real	The fourth default attribute for the set (only valid for Set.IGA_EDGE, Set.IGA_FACE, Set.IGA_POINT_UVW, Set.NODE, Set.PART, Set.SEGMENT, Set.SEGMENT_2D and Set.SHELL)
exists (read only)	logical	true if set exists, false if referred to but not defined.
general	logical	If _GENERAL option is active.
general_lines (read only)	integer	Number of lines of data for _GENERAL set (if _GENERAL option is active).
generate	logical	If _GENERATE option is active.
iga_opt (read only)	constant	Option for IGA_EGDE and IGA_FACE. Can be Set.IGA_UVW, Set.IGA_XYZ
include	integer	The Include file number that the set is in.
increment (read only)	logical	If _GENERATE_INCREMENT option is active.
intersect	logical	If _INTERSECT option is active.
its	integer	Coupling type across different scales in two-scale cosimulation (only valid for Set.SEGMENT or Set.NODE).
label	integer	Set number. Also see the sid property which is an alternative name for this.
model	integer	The Model number that the set is in.
sid	integer	Set number. Also see the label property which is an alternative name for this.
smooth (read only)	logical	If _LIST_SMOOTH option is active.
solver	string	Solver to attach to set. Can be "MECH", "CESE", "EM", "ICFD" or blank (only valid for Set.IGA_EDGE, Set.IGA_FACE, Set.IGA_POINT_UVW, Set.NODE, Set.PART, Set.SEGMENT and Set.SOLID).
title	string	Set title

total (read only)	integer	The total number of items in the set. Note that for _GENERAL and _GENERATE sets this is expensive to compute.
transparency	integer	The transparency of the set (0-100) 0% is opaque, 100% is transparent.
type (read only)	constant	Set type. Can be Set.BEAM, Set.BOX Set.DISCRETE, Set.IGA_EDGE, Set.IGA_FACE, Set.IGA_POINT_UVW, Set.MM_GROUP, Set.MODE, Set.NODE, Set.PART, Set.PART_TREE, Set.PERI_LAMINATE, Set.SEGMENT, Set.SEGMENT_2D, Set.SHELL, Set.SOLID or Set.TSHELL

Detailed Description

The Set class allows you to create, modify, edit and manipulate sets. See the documentation below for more details.
ECMAScript 6 defines a Set class for Set objects so unfortunately this clashes with the Set class we have defined in PRIMER for the Ansys LS-DYNA keyword *SET. By default the Set class is used for the Ansys LS-DYNA keyword *SET but this can be changed by using the preference 'set_class' in the preferences editor.
The Ansys LS-DYNA keyword *SET class is also available (regardless of whether Set is used for the keyword or ECMAScript 6 Set objects) by using SetK (similarly to Nrb being an alias for the NodalRigidBody class).

For convenience "SetK" can also be used as the class name instead of "Set".

Constructor

new Set(Model*[Model]*, details*[object]*)

Description

Create a new Set object.

Arguments

- **Model** (Model)

Model that set will be created in

- **details** (object)

Details for creating the Set

Object has the following properties:

Name	Type	Description
iga_opt (optional)	constant	IGA type. Can be Set.IGA_UVW or Set.IGA_XYZ. Only valid for set types Set.IGA_EDGE and Set.IGA_FACE where it must be given
option (optional)	constant	Set type. Can be Set.ADD, Set.INTERSECT, Set.GENERAL or Set.GENERATE
sid	integer	Set number
title (optional)	string	Title for the set
type	constant	Type of set. Can be Set.BEAM, Set.BOX, Set.DISCRETE, Set.IGA_EDGE, Set.IGA_FACE, Set.IGA_POINT_UVW, Set.MM_GROUP, Set.MODE, Set.NODE, Set.PART, Set.PART_TREE, Set.PERI_LAMINATE, Set.SEGMENT, Set.SEGMENT_2D, Set.SHELL, Set.SOLID or Set.TSHELL

Returns

Set object

Return type

Set

Example

To create a new node set in model m with label 100:

```
var s = new Set(m, {sid:100, type:Set.NODE});
```

To create a new *NODE_SET_ADD in model m with label 101:

```
var s = new Set(m, {sid:101, type:Set.NODE, title:"", option:Set.ADD});
```

To create a new *SET_IGA_EDGE_XYZ in model m with label 102:

```
var s = new Set(m, {sid:102, type:Set.IGA_EDGE, title:"Set of IGA_EDGE_XYZ",
iga_opt:Set.IGA_XYZ});
```

new Set(Model*[Model]*, sid*[integer]*, type*[constant]*, title (optional)*[string]*, option (optional)*[constant]*) [deprecated]

This function is deprecated in version 22.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Create a new Set object.

Arguments

- **Model** (Model)

Model that set will be created in

- **sid** (integer)

Set number

- **type** (constant)

Type of set. Can be Set.BEAM, Set.BOX, Set.DISCRETE, Set.MM_GROUP, Set.MODE, Set.NODE, Set.PART, Set.PART_TREE, Set.PERI_LAMINATE, Set.SEGMENT, Set.SEGMENT_2D, Set.SHELL, Set.SOLID or Set.TSHELL

- **title (optional)** (string)

Title for the set

- **option (optional)** (constant)

Set type. Can be Set.ADD, Set.INTERSECT, Set.GENERAL or Set.GENERATE

Returns

Set object

Return type

Set

Example

To create a new node set in model m with label 100:

```
var s = new Set(m, 100, Set.NODE);
```

To create a new *NODE_SET_ADD in model m with label 101:

```
var s = new Set(m, 101, Set.NODE,"", Set.ADD);
```

Details of functions

Add(id1*[integer]*, id2 (optional)*[integer]*, id3 (optional)*[integer]*, id4 (optional)*[integer]*)

Description

Adds an item to the set. **This cannot be used for _COLUMN and _GENERAL sets**. For segment sets four nodes must be given to define a segment to add to the set.

Arguments

- **id1** (integer)

id of the item to add to the set (normal, _ADD or _ADD_ADVANCED sets) or Start ID (_GENERATE sets)

- **id2 (optional)** (integer)

type of the item to add to the set [1-7] (_ADD_ADVANCED sets) or End ID (_GENERATE sets)
(only for SEGMENT, _GENERATE, _GENERATE_INCREMENT and _ADD_ADVANCED sets)

- **id3 (optional)** (integer)

Increment for _GENERATE_INCREMENT sets, otherwise id of the item to add to the set
(only for SEGMENT and _GENERATE_INCREMENT sets)

- **id4 (optional)** (integer)

id of the item to add to the set
(only for SEGMENT sets)

Returns

No return value

Example

To add node 10 to node set ns:

```
ns.Add(10);
```

To add segment 10, 11, 12, 13 to segment set ss:

```
ss.Add(10, 11, 12, 13);
```

SET PART TREE is a special type of set, which can contain PARTs (negative) and/or child SET PART TREEs (positive).
To add part 10 to SET PART TREE spt:

```
spt.Add(-10);
```

To add child SET PART TREE 20 to SET PART TREE spt:

```
spt.Add(20);
```

AddCollectChild(set*[Set]*)

Description

Adds a child collect set to the set. The child set label will be changed to be the same as the parent set and it will become a child. Also see Set.collect_children and Set.GetCollectChild.

Arguments

- **set** (Set)

Set to be added as a child collect set.

Returns

No return value

Example

To make set ns2 to node set ns:

```
ns.AddCollectChild(ns2);
```

AddFlagged(flag*[Flag]*)

Description

Adds flagged items to the set. **This cannot be used for _GENERAL or _GENERATE sets** and **cannot be used for segment sets**

Arguments

- **flag** (Flag)

Flag for items to add to the set

Returns

No return value

Example

To add any nodes flagged with flag f to node set ns:

```
ns.AddFlagged(f);
```

AllItems()

Description

Returns an array containing all of the items in the set, decomposing any complex set definitions as required to give those items. For Set.SEGMENT sets, each index in the array is an array containing the segment node IDs. For all other set types each index in the array is an item ID

Arguments

No arguments

Returns

array of numbers

Return type

Number

Example

To get the items in set s:

```
var items = s.AllItems();
```

BlankAll(Model*[Model]*, type (optional)*[constant]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the sets in the model.

Arguments

- **Model** (Model)

Model that all sets will be blanked in

- **type (optional)** (constant)

Type of sets to blank. Can be Set.BEAM, Set.BOX, Set.DISCRETE, Set.MM_GROUP, Set.NODE, Set.PART, Set.PART_TREE, Set.PERI_LAMINATE, Set.SEGMENT, Set.SEGMENT_2D, Set.SHELL, Set.SOLID or Set.TSHELL. Set.ALL_TYPES. If omitted sets of all types will be blanked.

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the sets in model m:

```
Set.BlankAll(m);
```

To blank all of the node sets in model m:

```
Set.BlankAll(m, Set.NODE);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, type (optional)*[constant]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged sets in the model.

Arguments

- **Model** (Model)

Model that all the flagged sets will be blanked in

- **flag** (Flag)

Flag set on the sets that you want to blank

- **type (optional)** (constant)

Type of sets to blank. Can be Set.BEAM, Set.BOX, Set.DISCRETE, Set.MM_GROUP, Set.NODE, Set.PART, Set.PART_TREE, Set.PERI_LAMINATE, Set.SEGMENT, Set.SEGMENT_2D, Set.SHELL, Set.SOLID or Set.TSHELL. Set.ALL_TYPES. If set, only flagged sets of this type will be blanked. If omitted flagged sets of all types will be blanked.

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the sets in model m flagged with f:

```
Set.BlankFlagged(m, f);
```

To blank all of the node sets in model m flagged with f:

```
Set.BlankFlagged(m, f, Set.NODE);
```

Blanked()

Description

Checks if the set is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if set s is blanked:

```
if (s.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

No return value

Example

To browse set s:

```
var s.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the set.

Arguments

- **flag** (Flag)

Flag to clear on the set

Returns

No return value

Example

To clear flag f for set s:

```
s.ClearFlag(f);
```

Contains(id*[integer]*)

Description

Checks if an item is in the set. **This cannot be used for ADD_ADVANCED, _GENERAL or _GENERATE sets** and **cannot be used for segment sets**

Arguments

- **id** (integer)

id of the item to check.

Returns

true if item is in set, false if not

Return type

Boolean

Example

To see if node 10 is in node set ns:

```
if (ns.Contains(10) )
{
    do something...
}
```

Copy(range (optional)*[boolean]*)

Description

Copies the set.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. To set current include, use Include.MakeCurrentLayer() .

Returns

Set object

Return type

Set

Example

To copy node net ns into node net ns1:

```
var ns1 = ns.Copy();
```

Create(Model*[Model]*, type*[constant]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a set.

Arguments

- **Model** (Model)

Model that the set will be created in

- **type** (constant)

Type of the set that you want to create. Can be Set.BEAM, Set.BOX, Set.DISCRETE, Set.IGA_EDGE, Set.IGA_FACE, Set.IGA_POINT_UVW, Set.MM_GROUP, Set.MODE, Set.NODE, Set.PART, Set.PART_TREE, Set.PERI_LAMINATE, Set.SEGMENT, Set.SEGMENT_2D, Set.SHELL, Set.SOLID or Set.TSHELL

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

Set object (or null if not made)

Return type

Set

Example

To start creating a node set in model m:

```
var s = Set.Create(m, Set.NODE);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel to edit the set.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

No return value

Example

To edit set s:

```
var s.Edit();
```

Empty()

Description

Removes all items from the set. **This cannot be used for _GENERATE sets** and **cannot be used for segment sets**

Arguments

No arguments

Returns

No return value

Example

To remove all nodes from node set ns:

```
ns.Empty(f);
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for a set with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The set object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for set s:

```
s.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for set s:

```
s.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for a set. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for set s:

```
s.Error("My custom error");
```

First(Model*[Model]*, type*[constant]*) [static]

Description

Returns the first set in the model.

Arguments

- **Model** (Model)

Model to get first set in

- **type** (constant)

Type of the set. Can be Set.BEAM, Set.BOX, Set.DISCRETE, Set.IGA_EDGE, Set.IGA_FACE, Set.IGA_POINT_UVW, Set.MM_GROUP, Set.MODE, Set.NODE, Set.PART, Set.PART_TREE, Set.PERI_LAMINATE, Set.SEGMENT, Set.SEGMENT_2D, Set.SHELL, Set.SOLID or Set.TSHELL

Returns

Set object (or null if there are no sets in the model).

Return type

Set

Example

To get the first node set in model m:

```
var n = Set.First(m, Set.NODE);
```

FirstFreeLabel(Model*[Model]*, type*[constant]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free set label in the model. Also see Set.LastFreeLabel(), Set.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free Set label in

- **type** (constant)

Type of the set. Can be Set.BEAM, Set.BOX, Set.DISCRETE, Set.IGA_EDGE, Set.IGA_FACE, Set.IGA_POINT_UVW, Set.MM_GROUP, Set.MODE, Set.NODE, Set.PART, Set.PART_TREE, Set.PERI_LAMINATE, Set.SEGMENT, Set.SEGMENT_2D, Set.SHELL, Set.SOLID or Set.TSHELL

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

Set label.

Return type

Number

Example

To get the first free node set label in model m:

```
var label = Set.FirstFreeLabel(m, Set.NODE);
```

FlagAll(Model*[Model]*, flag*[Flag]*, type (optional)*[constant]*) [static]

Description

Flags all of the sets in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all sets will be flagged in

- **flag** (Flag)

Flag to set on the sets

- **type (optional)** (constant)

Type of the set. Can be Set.BEAM, Set.BOX Set.DISCRETE, Set.IGA_EDGE, Set.IGA_FACE, Set.IGA_POINT_UVW, Set.MM_GROUP, Set.MODE, Set.NODE, Set.PART, Set.PART_TREE, Set.PERI_LAMINATE, Set.SEGMENT, Set.SEGMENT_2D, Set.SHELL, Set.SOLID or Set.TSHELL. If set, only sets of this type will be flagged. If omitted sets of all types will be flagged.

Returns

No return value

Example

To flag all of the node sets with flag f in model m:

```
Set.FlagAll(m, f, Set.NODE);
```

Flagged(flag*[Flag]*)

Description

Checks if the set is flagged or not.

Arguments

- **flag** (Flag)

Flag to clear on the set

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if set s has flag f set on it:

```
if (s.Flagged(f) ) do_something...
```

GetAll(Model*[Model]*, type*[constant]*) [static]

Description

Returns an array of Set objects for all of the sets in a models in PRIMER

Arguments

- **Model** (Model)

Model to get sets from

- **type** (constant)

Type of the set. Can be Set.BEAM, Set.BOX, Set.DISCRETE, Set.IGA_EDGE, Set.IGA_FACE, Set.IGA_POINT_UVW, Set.MM_GROUP, Set.MODE, Set.NODE, Set.PART, Set.PART_TREE, Set.PERI_LAMINATE, Set.SEGMENT, Set.SEGMENT_2D, Set.SHELL, Set.SOLID or Set.TSHELL

Returns

Array of Set objects

Return type

Array

Example

To make an array of Set objects for all of the node sets in model m

```
var n = Set.GetAll(m, Set.NODE);
```

GetCollectChild(number*[Integer]*)

Description

Returns a child collect set. Also see Set.collect_children and Set.AddCollectChild.

Arguments

- **number** (Integer)

The index of the child collect set to return. **Note that indices start at 0, not 1**

Returns

Set object

Return type

Set

Example

To loop over the child collect sets for set ns:

```
if (ns.collect)
{
    for (i=0; i<ns.collect_children; i++)
        var child = ns.GetCollectChild(i);
}
```

GetFlagged(Model*[Model]*, flag*[Flag]*, type*[constant]*) [static]

Description

Returns an array of Set objects for all of the flagged sets in a models in PRIMER

Arguments

- **Model** (Model)

Model to get sets from

- **flag** (Flag)

Flag set on the set that you want to retrieve

- **type** (constant)

Type of the set. Can be Set.BEAM, Set.BOX, Set.DISCRETE, Set.IGA_EDGE, Set.IGA_FACE, Set.IGA_POINT_UVW, Set.MM_GROUP, Set.MODE, Set.NODE, Set.PART, Set.PART_TREE, Set.PERI_LAMINATE, Set.SEGMENT, Set.SEGMENT_2D, Set.SHELL, Set.SOLID or Set.TSHELL

Returns

Array of Set objects

Return type

Array

Example

To make an array of Set objects for all of the node sets in model m flagged with f

```
var n = Set.GetFlagged(m, f, Set.NODE);
```

GetFromID(Model*[Model]*, set number*[integer]*, type*[constant]*) [static]

Description

Returns the Set object for a set ID.

Arguments

- **Model** (Model)

Model to find the set in

- **set number** (integer)

number of the set you want the Set object for

- **type** (constant)

Type of the set. Can be Set.BEAM, Set.BOX, Set.DISCRETE, Set.IGA_EDGE, Set.IGA_FACE, Set.IGA_POINT_UVW, Set.MM_GROUP, Set.MODE, Set.NODE, Set.PART, Set.PART_TREE, Set.PERI_LAMINATE, Set.SEGMENT, Set.SEGMENT_2D, Set.SHELL, Set.SOLID or Set.TSHELL

Returns

Set object (or null if set does not exist).

Return type

Set

Example

To get the Set object for node set 100 in model m

```
var n = Set.GetFromID(m, 100, Set.NODE);
```

GetGeneralData(index*[Integer]*)

Description

Returns a line of data for a GENERAL set.

Arguments

- **index** (Integer)

The index of the GENERAL data to return. **Note that indices start at 0, not 1**.
0 <= index < general_lines

Returns

Array containing data.

Return type

Array

Example

To loop over the lines of general data sets for set s:

```
if (s.general)
{
    for (i=0; i<s.general_lines; i++)
        var data = s.GetGeneralData(i);
}
```

Keyword()

Description

Returns the keyword for this set (*SET_NODE etc). **Note that a carriage return is not added**. See also Set.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for set s:

```
var key = s.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the set. **Note that a carriage return is not added**. See also Set.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for set s:

```
var cards = s.KeywordCards();
```

Last(Model*[Model]*, type*[constant]*) [static]

Description

Returns the last set in the model.

Arguments

- **Model** (Model)

Model to get last set in

- **type** (constant)

Type of the set. Can be Set.BEAM, Set.BOX, Set.DISCRETE, Set.IGA_EDGE, Set.IGA_FACE, Set.IGA_POINT_UVW, Set.MM_GROUP, Set.MODE, Set.NODE, Set.PART, Set.PART_TREE, Set.PERI_LAMINATE, Set.SEGMENT, Set.SEGMENT_2D, Set.SHELL, Set.SOLID or Set.TSHELL

Returns

Set object (or null if there are no sets in the model).

Return type

Set

Example

To get the last node set in model m:

```
var n = Set.Last(m, Set.NODE);
```

LastFreeLabel(Model*[Model]*, type*[constant]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free set label in the model. Also see Set.FirstFreeLabel(), Set.NextFreeLabel() and Model.LastFreeItemLabel().

Arguments

- **Model** (Model)

Model to get last free Set label in

- **type** (constant)

Type of the set. Can be Set.BEAM, Set.BOX, Set.DISCRETE, Set.IGA_EDGE, Set.IGA_FACE, Set.IGA_POINT_UVW, Set.MM_GROUP, Set.MODE, Set.NODE, Set.PART, Set.PART_TREE, Set.PERI_LAMINATE, Set.SEGMENT, Set.SEGMENT_2D, Set.SHELL, Set.SOLID or Set.TSHELL

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

Set label.

Return type

Number

Example

To get the last free node set label in model m:

```
var label = Set.LastFreeLabel(m, Set.NODE);
```

Next()

Description

Returns the next set in the model.

Arguments

No arguments

Returns

Set object (or null if there are no more sets in the model).

Return type

Set

Example

To get the set in model m after set n:

```
var n = n.Next();
```

NextFreeLabel(Model*[Model]*, type*[constant]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free set label in the model. Also see Set.FirstFreeLabel(), Set.LastFreeLabel() and Model.NextFreeItemLabel().

Arguments

- **Model** (Model)

Model to get next free Set label in

- **type** (constant)

Type of the set. Can be Set.BEAM, Set.BOX, Set.DISCRETE, Set.IGA_EDGE, Set.IGA_FACE, Set.IGA_POINT_UVW, Set.MM_GROUP, Set.MODE, Set.NODE, Set.PART, Set.PART_TREE, Set.PERI_LAMINATE, Set.SEGMENT, Set.SEGMENT_2D, Set.SHELL, Set.SOLID or Set.TSHELL

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

Set label.

Return type

Number

Example

To get the next free node set label in model m:

```
var label = Set.NextFreeLabel(m, Set.NODE);
```

Pick(type*[constant]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a set.

Arguments

- **type** (constant)

Type of sets to pick. Can be Set.BEAM, Set.BOX, Set.DISCRETE, Set.MM_GROUP, Set.NODE, Set.PART, Set.PART_TREE, Set.PERI_LAMINATE, Set.SEGMENT, Set.SEGMENT_2D, Set.SHELL, Set.SOLID or Set.TSHELL.

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only sets from that model can be picked. If the argument is a Flag then only sets that are flagged with *limit* can be selected. If omitted, or null, any sets from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

Set object (or null if not picked)

Return type

Set

Example

To pick a node set from model m giving the prompt 'Pick set from screen':

```
var n = Set.Pick(Set.NODE, 'Pick set from screen', m);
```

Previous()

Description

Returns the previous set in the model.

Arguments

No arguments

Returns

Set object (or null if there are no more sets in the model).

Return type

Set

Example

To get the set in model m before this one:

```
var s = s.Previous();
```

RebuildCache()

Description

Rebuilds the cache for a set. As sets can be built using complex combinations of _GENERAL, _ADD, _INTERSECT options etc PRIMER creates a 'cache' for the set to speed up set drawing and usage. During normal interactive use this cache is rebuilt as necessary but in JavaScript it is possible for the cache to become out of date (e.g. you change a box position in JavaScript that is used by a *SET_GENERAL). Calling this forces the cache to be rebuilt.

Arguments

No arguments

Returns

No return type

Return type

null

Example

To rebuild the cache for set s:

```
s.RebuildCache();
```

Remove(id*[integer]*)

Description

Removes an item from the set. If the item is not in the set nothing is done. **This cannot be used for ADD_ ADVANCED, _COLUMN, _GENERAL or _GENERATE sets** and **cannot be used for segment sets**

Arguments

- **id** (integer)

id of the item to remove from the set.

Returns

No return value

Example

To remove node 10 from node set ns:

```
ns.Remove(10);
```

RemoveFlagged(flag*[Flag]*)

Description

Removes flagged items from the set. **This cannot be used for _GENERAL or _GENERATE sets** and **cannot be used for segment sets**

Arguments

- **flag** (Flag)

Flag for items to remove from the set

Returns

No return value

Example

To remove any nodes flagged with flag f from node set ns:

```
ns.RemoveFlagged(f);
```

RemoveGeneralData(index*[Integer]*)

Description

Removes a line of data from a GENERAL set.

Arguments

- **index** (Integer)

The index of the GENERAL data to remove. **Note that indices start at 0, not 1**.
0 <= index < general_lines

Returns

No return value

Example

To remove the first line of general data sets for set s:

```
if (s.general)
{
    s.RemoveGeneralData(0);
}
```

RenumberAll(Model*[Model]*, start*[integer]*, type (optional)*[constant]*) [static]

Description

Renumbers all of the sets in the model.

Arguments

- **Model** (Model)

Model that all sets will be renumbered in

- **start** (integer)

Start point for renumbering

- **type (optional)** (constant)

Type of sets to renumber. Can be Set.BEAM, Set.BOX Set.DISCRETE, Set.IGA_EDGE, Set.IGA_FACE, Set.IGA_POINT_UVW, Set.MM_GROUP, Set.MODE, Set.NODE, Set.PART, Set.PART_TREE, Set.PERI_LAMINATE,

Set.SEGMENT, Set.SEGMENT_2D, Set.SHELL, Set.SOLID or Set.TSHELL. If omitted sets of all types will be blanked.

Returns

No return value

Example

To renumber all of the sets in model m, from 1000000:

```
Set.RenumberAll(m, 1000000);
```

To renumber all of the node sets in model m, from 1000000:

```
Set.RenumberAll(m, 1000000, Set.NODE);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*, type (optional)*[constant]*) [static]

Description

Renumbers all of the flagged sets in the model.

Arguments

- **Model** (Model)

Model that all the flagged sets will be renumbered in

- **flag** (Flag)

Flag set on the sets that you want to renumber

- **start** (integer)

Start point for renumbering

- **type (optional)** (constant)

Type of sets to renumber. Can be Set.BEAM, Set.BOX Set.DISCRETE, Set.IGA_EDGE, Set.IGA_FACE, Set.IGA_POINT_UVW, Set.MM_GROUP, Set.MODE, Set.NODE, Set.PART, Set.PART_TREE, Set.PERI_LAMINATE, Set.SEGMENT, Set.SEGMENT_2D, Set.SHELL, Set.SOLID or Set.TSHELL. If omitted sets of all types will be blanked.

Returns

No return value

Example

To renumber all of the sets in model m flagged with f, from 1000000:

```
Set.RenumberFlagged(m, f, 1000000);
```

To renumber all of the node sets in model m flagged with f, from 1000000:

```
Set.RenumberFlagged(m, f, 1000000, Set.NODE);
```

Select(type*[constant]*, flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select sets using standard PRIMER object menus.

Arguments

- **type** (constant)

Type of sets to pick. Can be Set.BEAM, Set.BOX Set.DISCRETE, Set.IGA_EDGE, Set.IGA_FACE, Set.IGA_POINT_

UVW, Set.MM_GROUP, Set.MODE, Set.NODE, Set.PART, Set.PART_TREE, Set.PERI_LAMINATE, Set.SEGMENT, Set.SEGMENT_2D, Set.SHELL, Set.SOLID or Set.TSHELL.

- **flag** (Flag)

Flag to use when selecting sets

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only sets from that model can be selected. If the argument is a Flag then only sets that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any sets from any model can be selected.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of items selected or null if menu cancelled

Return type

Number

Example

To select node sets from model m, flagging those selected with flag f, giving the prompt 'Select sets':

```
Set.Select(Set.NODE, f, 'Select sets', m);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the set.

Arguments

- **flag** (Flag)

Flag to set on the set

Returns

No return value

Example

To set flag f for set s:

```
s.SetFlag(f);
```

SetGeneralData(index*[Integer]*, data*[Array of data]*)

Description

Sets a line of data for a GENERAL set.

Arguments

- **index** (Integer)

The index of the GENERAL data to set. **Note that indices start at 0, not 1**.
0 <= index <= general_lines

- **data** (Array of data)

Array containing GENERAL data to set.

Returns

No return value.

Example

To add nodes inside boxes 1, 2 and 3 as a new line of data to node general set s:

```
var data = [ "BOX", 1, 2, 3];
s.SetGeneralData(s.general_lines, data);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the set. The set will be sketched until you either call Set.Unsketch(), Set.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the set is sketched. If omitted redraw is true. If you want to sketch several sets and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch set s:

```
s.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, type (optional)*[constant]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged sets in the model.

Arguments

- **Model** (Model)

Model that all the flagged sets will be sketched in

- **flag** (Flag)

Flag set on the sets that you want to sketch

- **type (optional)** (constant)

Type of sets to sketch. Can be Set.BEAM, Set.BOX Set.DISCRETE, Set.MM_GROUP, Set.NODE, Set.PART, Set.PART_TREE, Set.PERI_LAMINATE, Set.SEGMENT, Set.SEGMENT_2D, Set.SHELL, Set.SOLID or Set.TSHELL. Set.ALL_TYPES. If set, only flagged sets of this type will be sketched. If omitted flagged sets of all types will be sketched.

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is true. If you want to do several (un)sketches and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To sketch all of the sets in model m flagged with f:

```
Set.SketchFlagged(m, f);
```

To sketch all of the node sets in model m flagged with f:

```
Set.SketchFlagged(m, f, Set.NODE);
```

Spool()

Description

Spools a set, entry by entry. See also Set.StartSpool

Arguments

No arguments

Returns

For Set.SEGMENT returns an array containing node IDs, for all other set types returns the ID of the item. Returns 0 if no more items

Return type

Array

Example

To spool set s:

```
var id;
s.StartSpool();
while (id = s.Spool())
{
    do something...
}
```

StartSpool(raw (optional)*[boolean]*)

Description

Starts a set spooling operation. See also Set.Spool

Arguments

- **raw (optional)** (boolean)

If true then the raw data from _GENERATE, _ADD and _INTERSECT sets will be returned instead of expanding the data ranges or child set contents. If omitted raw will be false.

Returns

No return value

Example

To start spooling set s:

```
s.StartSpool();
```

UnblankAll(Model*[Model]*, type (optional)*[constant]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the sets in the model.

Arguments

- **Model** (Model)

Model that all sets will be unblanked in

- **type (optional)** (constant)

Type of sets to unblank. Can be Set.BEAM, Set.BOX Set.DISCRETE, Set.MM_GROUP, Set.NODE, Set.PART, Set.PART_TREE, Set.PERI_LAMINATE, Set.SEGMENT, Set.SEGMENT_2D, Set.SHELL, Set.SOLID or Set.TSHELL. Set.ALL_TYPES. If omitted sets of all types will be blanked.

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the sets in model m:

```
Set.UnblankAll(m);
```

To unblank all of the node sets in model m:

```
Set.UnblankAll(m, Set.NODE);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, type (optional)*[constant]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged sets in the model.

Arguments

- **Model** (Model)

Model that the flagged sets will be unblanked in

- **flag** (Flag)

Flag set on the sets that you want to unblank

- **type (optional)** (constant)

Type of sets to unblank. Can be Set.BEAM, Set.BOX Set.DISCRETE, Set.MM_GROUP, Set.NODE, Set.PART, Set.PART_TREE, Set.PERI_LAMINATE, Set.SEGMENT, Set.SEGMENT_2D, Set.SHELL, Set.SOLID or Set.TSHELL. Set.ALL_TYPES. If set, only flagged sets of this type will be unblanked. If omitted flagged sets of all types will be unblanked.

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the sets in model m flagged with f:

```
Set.UnblankFlagged(m, f);
```

To unblank all of the node sets in model m flagged with f:

```
Set.UnblankFlagged(m, f, Set.NODE);
```

UnflagAll(Model*[Model]*, flag*[Flag]*, type (optional)*[constant]*) [static]

Description

Unsets a defined flag on all of the sets in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all sets will be unset in

- **flag** (Flag)

Flag to unset on the sets

- **type (optional)** (constant)

Type of the set. Can be Set.BEAM, Set.BOX Set.DISCRETE, Set.IGA_EDGE, Set.IGA_FACE, Set.IGA_POINT_UVW, Set.MM_GROUP, Set.MODE, Set.NODE, Set.PART, Set.PART_TREE, Set.PERI_LAMINATE, Set.SEGMENT, Set.SEGMENT_2D, Set.SHELL, Set.SOLID or Set.TSHELL

Returns

No return value

Example

To unset the flag f on all the sets in model m:

```
Set.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the set.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the set is unsketched. If omitted redraw is true. If you want to unsketch several sets and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch set s:

```
s.Unsketch();
```

UnsketchAll(Model*[Model]*, type (optional)*[constant]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all sets.

Arguments

- **Model** (Model)

Model that all sets will be unsketched in

- **type (optional)** (constant)

Type of sets to unsketch. Can be Set.BEAM, Set.BOX Set.DISCRETE, Set.MM_GROUP, Set.NODE, Set.PART, Set.PART_TREE, Set.PERI_LAMINATE, Set.SEGMENT, Set.SEGMENT_2D, Set.SHELL, Set.SOLID or Set.TSHELL. If omitted sets of all types will be unsketched.

- **redraw (optional)** (boolean)

If model should be redrawn or not after the sets are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all sets in model m:

```
Set.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, type (optional)*[constant]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged sets.

Arguments

- **Model** (Model)

Model that all sets will be unsketched in

- **flag** (Flag)

Flag set on the sets that you want to unsketch

- **type (optional)** (constant)

Type of sets to unsketch. Can be Set.BEAM, Set.BOX Set.DISCRETE, Set.MM_GROUP, Set.NODE, Set.PART, Set.PART_TREE, Set.PERI_LAMINATE, Set.SEGMENT, Set.SEGMENT_2D, Set.SHELL, Set.SOLID or Set.TSHELL. If omitted sets of all types will be unsketched.

- **redraw (optional)** (boolean)

If model should be redrawn or not after the sets are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all sets in model m flagged with f:

```
Set.UnsketchFlagged(m, f);
```

To unsketch all of the node sets in model m flagged with f:

```
Set.UnsketchFlagged(m, f, Set.NODE);
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for a set with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The set object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for set s:

```
s.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for set s:

```
s.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for a set. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for set s:

```
s.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this set.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for this set:

```
var xrefs = s.Xrefs();
```

---

toString()

Description

Creates a string containing the set data in keyword format. Note that this contains the keyword header and the keyword cards. See also Set.Keyword() and Set.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for set n in keyword format

```
var s = n.toString();
```

Termination class

The Termination class gives you access to *TERMINATION_XXXX cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

Termination constants

Constants for Direction for Degrees of Freedom (field: dof)

Name	Description
Termination.DOF_X	Degree of freedom in X direction for Force magnitude. Used for Termination.CONTACT.
Termination.DOF_Y	Degree of freedom in Y direction for Force magnitude. Used for Termination.CONTACT.
Termination.DOF_Z	Degree of freedom in Z direction for Force magnitude. Used for Termination.CONTACT.

Constants for Global Direction for Stop Criterion (field: stop)

Name	Description
Termination.STOP_MAG	Stop if displacement magnitude is exceeded. Used for Termination.BODY or Termination.NODE.
Termination.STOP_X	Stop criterion is in Global X direction. Used for Termination.BODY or Termination.NODE.
Termination.STOP_Y	Stop criterion is in Global Y direction. Used for Termination.BODY or Termination.NODE.
Termination.STOP_Z	Stop criterion is in Global X direction. Used for Termination.BODY or Termination.NODE.

Constants for Type of Keyword

Name	Description
Termination.BODY	TERMINATION is *TERMINATION_BODY.
Termination.CONTACT	TERMINATION is *TERMINATION_CONTACT.
Termination.CURVE	TERMINATION is *TERMINATION_CURVE.
Termination.DELETED_SHELLS	TERMINATION is *TERMINATION_DELETED_SHELLS.
Termination.DELETED_SHELLS_SET	TERMINATION is *TERMINATION_DELETED_SHELLS_SET.
Termination.DELETED_SOLIDS	TERMINATION is *TERMINATION_DELETED_SOLIDS.
Termination.DELETED_SOLIDS_SET	TERMINATION is *TERMINATION_DELETED_SOLIDS_SET.
Termination.NODE	TERMINATION is *TERMINATION_NODE.
Termination.SENSOR	TERMINATION is *TERMINATION_SENSOR.

Termination properties

Name	Type	Description
actTime	real	Activation time value. Used for Termination.CONTACT or Termination.CURVE.
dof	integer	Directions to consider for Force Magnitude. Valid values are: Termination.DOF_X or Termination.DOF_Y or Termination.DOF_Z. Used for Termination.CONTACT.
duration	real	Time duration of null resultant force prior to termination. Used for Termination.CONTACT.
exists (read only)	logical	true if termination exists, false if referred to but not defined.
id	integer	Can be Part or NRBC ID based on ptype value for Termination.BODY, OR Contact ID for Termination.CONTACT, OR Node ID for Termination.NODE, OR Curve ID for Termination.CURVE, OR Part for Termination.DELETED_SHELLS or Termination.DELETED_SOLIDS, OR Part Set ID for Termination.DELETED_SHELLS_SET or Termination.DELETED_SOLIDS_SET, OR Sensor Switch ID for Termination.SENSOR.
include	integer	The Include file number that the termination is in.
maxc	real	Maximum (most positive) displacement. If value is 0.0, it is set to 1.0e21. Value should be more than minc. Used for Termination.BODY or Termination.NODE.
minc	real	Minimum (most negative) displacement. If value is 0.0, it is set to -1.0e21. Value should be less than maxc. Used for Termination.BODY or Termination.NODE.
model (read only)	integer	The Model number that the termination is in.
numDeletedElems	integer	Number of elements that must be deleted for the specified Part ID's, before an error termination occurs. Used for Termination.DELETED_SHELLS_SET or Termination.DELETED_SOLIDS_SET.

ptype (read only)	integer	Gives the type of Part for Termination.BODY. Values can be 0 for Part or 1 for NRBC
stop	integer	Stop Criterion. Valid values are: Termination.STOP_X or Termination.STOP_Y or Termination.STOP_Z or Termination.STOP_MAG. Used for Termination.BODY or Termination.NODE.
threshold	real	Any measured force magnitude below or equal to this specified threshold is taken as a null force. Used for Termination.CONTACT.
type (read only)	integer	Gives the type of *TERMINATION keyword.

Detailed Description

The Termination class allows you to create, modify, edit and manipulate termination cards. See the documentation below for more details.

Constructor

new Termination(Model*[Model]*, Type*[constant]*, id *[integer]*)

Description

Create a new Termination object.

Arguments

- **Model** (Model)

Model that termination will be created in

- **Type** (constant)

Specify the type of Termination (Can be Termination.BODY or Termination.CONTACT or Termination.CURVE or Termination.DELETED_SHELLS or Termination.DELETED_SOLIDS or Termination.NODE or Termination.SENSOR).

- **id** (integer)

Can be Part ID for Termination.BODY or Termination.DELETED_SHELLS or Termination.DELETED_SOLIDS, OR Contact ID for Termination.CONTACT, OR Node ID for Termination.NODE, OR Curve ID for Termination.CURVE, OR Part Set ID for Termination.DELETED_SHELLS_SET or Termination.DELETED_SOLIDS_SET, OR Sensor Switch ID for Termination.SENSOR.

Returns

Termination object

Return type

Termination

Example

To create a new termination in model m, type BODY, part id 100:

```
var term = new Termination(m, Termination.BODY, 100);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a termination.

Arguments

- **Comment** (Comment)

Comment that will be attached to the termination

Returns

No return value

Example

To associate comment c to the termination term:

```
term.AssociateComment(c);
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse termination term:

```
term.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the termination.

Arguments

- **flag** (Flag)

Flag to clear on the termination

Returns

No return value

Example

To clear flag f for termination term:

```
term.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the termination. The target include of the copied termination can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current

include, use Include.MakeCurrentLayer().

Returns

Termination object

Return type

Termination

Example

To copy termination term into termination z:

```
var z = term.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a termination

Arguments

- **Model** (Model)

Model that the termination will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

Termination object (or null if not made).

Return type

Termination

Example

To start creating a termination in model m:

```
var term = Termination.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a termination.

Arguments

- **Comment** (Comment)

Comment that will be detached from the termination

Returns

No return value

Example

To detach comment c from the termination term:

```
term.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit termination term:

```
term.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for termination with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The termination object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for termination term:

```
term.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for termination term:

```
term.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for termination. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for termination term:

```
term.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first termination in the model.

Arguments

- **Model** (Model)

Model to get first termination in

Returns

Termination object (or null if there are no terminations in the model).

Return type

Termination

Example

To get the first termination in model m:

```
var term = Termination.First(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the terminations in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all terminations will be flagged in

- **flag** (Flag)

Flag to set on the terminations

Returns

No return value

Example

To flag all of the terminations with flag f in model m:

```
Termination.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the termination is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the termination

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if termination term has flag f set on it:

```
if (term.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each termination in the model.
Note that ForEach has been designed to make looping over terminations as fast as possible and so has some limitations.
Firstly, a single temporary Termination object is created and on each function call it is updated with the current termination data. This means that you should not try to store the Termination object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new terminations inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all terminations are in

- **func** (function)

Function to call for each termination

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the terminations in model m:

```
Termination.ForEach(m, test);
function test(term)
{
// term is Termination object
}
```

To call function test for all of the terminations in model m with optional object:

```
var data = { x:0, y:0 };
Termination.ForEach(m, test, data);
function test(term, extra)
{
// term is Termination object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Termination objects or properties for all of the terminations in a model in PRIMER. If the optional property argument is not given then an array of Termination objects is returned. If the property argument is given, that property value for each termination is returned in the array instead of a Termination object

Arguments

- **Model** (Model)

Model to get terminations from

- **property (optional)** (string)

Name for property to get for all terminations in the model

Returns

Array of Termination objects or properties

Return type

Array

Example

To make an array of Termination objects for all of the terminations in model m:

```
var a = Termination.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each termination in model m:

```
var a = Termination.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a termination.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the termination term:

```
var comm_array = term.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of Termination objects for all of the flagged terminations in a model in PRIMER If the optional property argument is not given then an array of Termination objects is returned. If the property argument is given, then that property value for each termination is returned in the array instead of a Termination object

Arguments

- **Model** (Model)

Model to get terminations from

- **flag** (Flag)

Flag set on the terminations that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged terminations in the model

Returns

Array of Termination objects or properties

Return type

Array

Example

To make an array of Termination objects for all of the terminations in model m flagged with f:

```
var term = Termination.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the terminations in model m flagged with f:

```
var a = Termination.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the Termination object for a termination ID.

Arguments

- **Model** (Model)

Model to find the termination in

- **number** (integer)

number of the termination you want the Termination object for

Returns

Termination object (or null if termination does not exist).

Return type

Termination

Example

To get the Termination object for termination 100 in model m

```
var term = Termination.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a Termination property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the Termination.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

termination property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if Termination property term.example is a parameter:

```
Options.property_parameter_names = true;
if (term.GetParameter(term.example) ) do_something...
Options.property_parameter_names = false;
```

To check if Termination property term.example is a parameter by using the GetParameter method:

```
if (term.ViewParameters().GetParameter(term.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this Termination (*TERMINATION_xxxx) **Note that a carriage return is not added**. See also Termination.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for Termination termination:

```
var key = Termination.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the Termination. **Note that a carriage return is not added**. See also Termination.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for Termination termination:

```
var cards = Termination.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last termination in the model.

Arguments

- **Model** (Model)

Model to get last termination in

Returns

Termination object (or null if there are no terminations in the model).

Return type

Termination

Example

To get the last termination in model m:

```
var term = Termination.Last(m);
```

Next()

Description

Returns the next termination in the model.

Arguments

No arguments

Returns

Termination object (or null if there are no more terminations in the model).

Return type

Termination

Example

To get the termination in model m after termination term:

```
var term = term.Next();
```

Previous()

Description

Returns the previous termination in the model.

Arguments

No arguments

Returns

Termination object (or null if there are no more terminations in the model).

Return type

Termination

Example

To get the termination in model m before termination term:

```
var term = term.Previous();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select terminations using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting terminations

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only terminations from that model can be selected. If the argument is a Flag then only terminations that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any terminations can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of terminations selected or null if menu cancelled

Return type

Number

Example

To select terminations from model m, flagging those selected with flag f, giving the prompt 'Select terminations':

```
Termination.Select(f, 'Select terminations', m);
```

To select terminations, flagging those selected with flag f but limiting selection to terminations flagged with flag l, giving the prompt 'Select terminations':

```
Termination.Select(f, 'Select terminations', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the termination.

Arguments

- **flag** (Flag)

Flag to set on the termination

Returns

No return value

Example

To set flag f for termination term:

```
term.SetFlag(f);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of terminations in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing terminations should be counted. If false or omitted referenced but undefined terminations will also be included in the total.

Returns

number of terminations

Return type

Number

Example

To get the total number of terminations in model m:

```
var total = Termination.Total(m);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the terminations in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all terminations will be unset in

- **flag** (Flag)

Flag to unset on the terminations

Returns

No return value

Example

To unset the flag f on all the terminations in model m:

```
Termination.UnflagAll(m, f);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

Termination object.

Return type

Termination

Example

To check if Termination property term.example is a parameter by using the Termination.GetParameter() method:

```
if (term.ViewParameters().GetParameter(term.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for termination with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The termination object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for termination term:

```
term.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for termination term:

```
term.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for termination. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for termination term:

```
term.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this termination.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for termination term:

```
var xrefs = term.Xrefs();
```

---

toString()

Description

Creates a string containing the Termination data in keyword format. Note that this contains the keyword header and the keyword cards. See also Termination.Keyword() and Termination.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for Termination termination in keyword format

```
var term = termination.toString();
```

Attached class

The Attached class contains constants and static functions relating to the Attached() member function from the Model class. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Beam3rdNodes(Setting*[boolean]*)
- BeamPid(Setting*[boolean]*)
- Deformable(Setting*[constant]*)
- FlagPart(Setting*[boolean]*)
- Recursive(Setting*[boolean]*, Iteration limit (optional)*[integer]*, Timeout (optional)*[integer]*)
- Rigid(Setting*[constant]*)
- SetEntity(Type*[string]*, Setting*[boolean]*)
- TiedContacts(Setting*[boolean]*)

Attached constants

Name	Description
Attached.SINGLE	Find attached option - find attached through single elements only
Attached.WHOLE	Find attached option - find through whole attached part

Detailed Description

The Attached class static functions are used to set options for the find attached feature in PRIMER. Once set, these settings are used when using the Attached member function from the Model class

Details of functions

Beam3rdNodes(Setting*[boolean]*) [static]

Description

Sets the find attached option for beam 3rd nodes on or off

Arguments

- **Setting** (boolean)

If true beam 3rd nodes are considered for find attached, if false, they are not.

Returns

No return value

Example

To set the 3rd node option to on:

```
Attached.Beam3rdNodes(true);
```

BeamPid(Setting*[boolean]*) [static]

Description

Sets the find attached option for beam pid on or off

Arguments

- **Setting** (boolean)

If true beam pid's are considered for find attached, if false, they are not.

Returns

No return value

Example

To set the beam pid option to on:

```
Attached.BeamPid(true);
```

Deformable(Setting*[constant]*) [static]

Description

Sets the deformable option for find attached

Arguments

- **Setting** (constant)

Option. Can be Attached.WHOLE, Attached.SINGLE

Returns

No return value

Example

To set the deformable option to find attached through the whole part:

```
Attached.Deformable(Attached.WHOLE);
```

FlagPart(Setting*[boolean]*) [static]

Description

Sets an option to flag parts after a find attached if any elements within that part are flagged

Arguments

- **Setting** (boolean)

If true, parts are flagged after a find attached if any elements within that part are flagged, if false, they are not.

Returns

No return value

Example

To set the flag part option to on:

```
Attached.FlagPart(true);
```

Recursive(Setting*[boolean]*, Iteration limit (optional)*[integer]*, Timeout (optional)*[integer]*) [static]

Description

Sets the find attached option for recursive on or off

Arguments

- **Setting** (boolean)

If true recursive is on, if false, it is off.

- **Iteration limit (optional)** (integer)

Limit eq 0 sets unlimited iterations. Limit gt 0 sets fixed number of iterations.

- **Timeout (optional)** (integer)

Sets timeout in seconds (default 60s) for recursive attached.

Returns

No return value

Example

To set the recursive option to run unlimited with 30s timeout

```
Attached.Recursive(true, 0, 30);
```

Rigid(Setting*[constant]*) [static]

Description

Sets the rigid option for find attached

Arguments

- **Setting** (constant)

Option. Can be Attached.WHOLE, Attached.SINGLE

Returns

No return value

Example

To set the rigid option to find attached through the whole part:

```
Attached.Rigid(Attached.WHOLE);
```

SetEntity(Type*[string]*, Setting*[boolean]*) [static]

Description

Sets entity to be on or off to find attached through.

Arguments

- **Type** (string)

The type of the item to switch on or off (for a list of types see Appendix A of the PRIMER manual). Use "ALL" to switch all entities or "CONSTRAINEDALL" to switch all constrained entities.

- **Setting** (boolean)

If true you turn the entity switch on, if false you turn it off.

Returns

No return value

Example

To set the SHELL switch to on so that when you run a find attached you fnd attached through shells:

```
Attached.SetEntity("SHELL", true);
```

TiedContacts(Setting*[boolean]*) [static]

Description

Sets the find attached option for tied contacts on or off

Arguments

- **Setting** (boolean)

If true tied contacts are considered for find attached, if false, they are not.

Returns

No return value

Example

To set the tied contacts option to on:

```
Attached.TiedContacts(true);
```

Belt class

The Belt class gives you access to belt fitting in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SetMeshingLabels(entity_type*[constant]*, label_value*[integer]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Fit()
- Flagged(flag*[Flag]*)
- Generate()
- GetComments()
- GetMesh(index*[integer]*)
- GetParameter(prop*[string]*)
- GetPoint(index*[integer]*)
- InsertPoint(index*[integer]*, position*[integer]*, data*[object]*)
- Next()
- Previous()
- RemovePoint(index*[integer]*)
- SetFlag(flag*[Flag]*)
- SetMesh(index*[integer]*, data*[object]*)
- SetPoint(index*[integer]*, data*[object]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()

- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()

Belt constants

Constants for Mesh segments

Name	Description
Belt.MSEG_B1_ONLY	Old style all 1D belt
Belt.MSEG_B2_ONLY	Old style all 2D belt
Belt.MSEG_BD_NEW	Indicates new mode. **This must be set before any of the new style constants can be used**
Belt.MSEG_CE_1D	New style 1D at centre
Belt.MSEG_CE_2D	New style 2D at centre
Belt.MSEG_CE_SH	New style SH at centre
Belt.MSEG_E1_1D	New style 1D at end 1
Belt.MSEG_E1_2D	New style 2D at end 1
Belt.MSEG_E1_SH	New style shells at end 1
Belt.MSEG_E2_1D	New style 1D at end 2
Belt.MSEG_E2_2D	New style 2D at end 2
Belt.MSEG_E2_SH	New style shells at end 2
Belt.MSEG_MIX_SB1	Old style 1D at ends, shells in middle
Belt.MSEG_MIX_SB2	Old style 2D at ends, shells in middle
Belt.MSEG_SH_ONLY	Old style all shell belt

Constants for Meshing start Labels

Name	Description
Belt.MESH_2D_SLIPRING_SET_NODE	Set meshing start Labels for 2D slipring node sets
Belt.MESH_ALL	Set meshing start Labels for everything used in the seatbelt definition
Belt.MESH_NODE	Set meshing start Labels for nodes
Belt.MESH_NRBC	Set meshing start Labels for nodal rigid bodies
Belt.MESH_RETRACTOR	Set meshing start Labels for retractors
Belt.MESH_SEATBELT	Set meshing start Labels for seatbelt elements
Belt.MESH_SET_NODE	Set meshing start Labels for node sets
Belt.MESH_SET_PART	Set meshing start Labels for part sets
Belt.MESH_SET_SHELL	Set meshing start Labels for shell sets
Belt.MESH_SHELL	Set meshing start Labels for shells
Belt.MESH_SLIPRING	Set meshing start Labels for slipring elements
Belt.MESH_XSEC	Set meshing start Labels for Database cross sections

Constants for Path point fixity

Name	Description
Belt.B_POST_ SLIPRING	There is a B-Post slipring at this point.
Belt.FIXED	Point is fixed
Belt.FREE_SLIPRING	There is a free (eg pelvis) slipring at this point.
Belt.KNOWN	The belt path is known to pass through this point
Belt.RETRACTOR	There is a retractor at this point
Belt.SLIPRING	There is a slipring at this point. (Deprecated from V12 onwards, use FREE_SLIPRING or B_POST_SLIPRING instead)
Belt.TWIST	Point has twist vectors or twist nodes defined
Belt.XSEC	There is a database cross section at this point

Constants for Path point insertion

Name	Description
Belt.INSERT_AFTER	Insert after given path point.
Belt.INSERT_BEFORE	Insert before given path point.

Belt properties

Name	Type	Description
acuteAngle	real	Limiting angle to be considered "acute" (0 means 90)
curvature	real	Maximum permitted transverse belt curvature in degrees
elemSet (read only)	integer	Set of shell or 2D seatbelt elements. Only created if the option to generate a contact for the belt is used
exists (read only)	logical	true if belt exists, false if referred to but not defined.
friction	real	Transverse friction coefficient
id	integer	Belt number. Also see the label property which is an alternative name for this.
include	integer	The Include file number that the belt is in.
iterations	integer	The number of fitting iterations between contact bucket resorts
label	integer	Belt number. Also see the id property which is an alternative name for this.
length	real	The characteristic length of each belt element
meshSegs (read only)	integer	Number of mesh segments defined
model (read only)	integer	The Model number that the belt is in.
n2sContact (read only)	integer	Nodes to Surface contact used between nodes on 1D belt elements and dummy structure. Only set if the AUTOMATIC_NODES_TO_SURFACE contact is defined in Seatbelts contact panel. Optional contact, see "Contact: Creating a Contact between Belt and Dummy" section in the PRIMER manual for more information.
nodeSet (read only)	integer	Set of all nodes in seatbelt. Only created if the option to generate a contact for the belt is used
nrbFirst (read only)	integer	First nodal rigid body ID

nrbLast (read only)	integer	Last nodal rigid body ID
nsboSet (read only)	integer	Set of nodes on 1D seatbelt elements only. Only created if the option to generate a contact for the belt is used
overlap	real	Fraction by which facets are extended during contact checking to stop nodes "falling into gaps"
parts	integer	Part set ID defining structure. Note that if you are creating the seatbelt definition from scratch in JavaScript you **must** define a shell, solid or thick shell set.
penetration	real	Maximum penetration distance considered for contact into solid and thick shell elements
pidShell	integer	The part ID for any 2D seatbelt elements
pid_1d	integer	The part ID for any 1D seatbelt elements
pid_2d	integer	The part ID for any 2D seatbelt elements
points (read only)	integer	Number of path points defined
projection	real	Initial projection distance by which belt path is "thrown outwards" at start of fitting
psiShell	real	Optional orthotropic angle for any shell elements
psi_2d	real	Optional orthotropic angle for any 2D seatbelt elements
retractorFirst (read only)	integer	First retractor ID
retractorLast (read only)	integer	Last retractor ID
rows	integer	The number of rows of 2D elements across the belt
s2sContact (read only)	integer	Surface to Surface contact used between shell/2D belt elements and dummy structure. Only set if the AUTOMATIC_SURFACE_TO_SURFACE contact is defined in Seatbelts contact panel. Optional contact, see "Contact: Creating a Contact between Belt and Dummy" section in the PRIMER manual for more information.
seatbeltFirst (read only)	integer	First 1D seatbelt ID
seatbeltLast (read only)	integer	Last 1D seatbelt ID
segments (read only)	integer	Segment set created for contact
shells	integer	Shell set ID defining structure. Note that if you are creating the seatbelt definition from scratch in JavaScript you **must** define a shell, solid or thick shell set.
slen_1d	real	The initial slack length for any 1D seatbelt elements
slipringFirst (read only)	integer	First slipring ID
slipringLast (read only)	integer	Last slipring ID
solids	integer	Solid set ID defining structure. Note that if you are creating the seatbelt definition from scratch in JavaScript you **must** define a shell, solid or thick shell set.
t1Shell	real	Optional thickness at n1 for any shell elements
t1_2d	real	Optional thickness at n1 for any 2D seatbelt elements
t2Shell	real	Optional thickness at n2 for any shell elements
t2_2d	real	Optional thickness at n2 for any 2D seatbelt elements
t3Shell	real	Optional thickness at n3 for any shell elements

t3_2d	real	Optional thickness at n3 for any 2D seatbelt elements
t4Shell	real	Optional thickness at n4 for any shell elements
t4_2d	real	Optional thickness at n4 for any 2D seatbelt elements
thickFactor	real	Factor used when thickFlag is 1
thickFlag	integer	Thickness used during fitting: 0 (default)=use true thickness; 1=use true thickness x factor; 2=use neutral axis (no thickness)
thickness	real	The thickness of 2D belt elements
title	string	Belt title.
tolerance	real	The convergence tolerance at which fitting halts
tshells	integer	Thick shell set ID defining structure. Note that if you are creating the seatbelt definition from scratch in JavaScript you **must** define a shell, solid or thick shell set.
width	real	The overall belt width
xsect_pretext	string	If X-Section pretext option is set to 2 then string for additional pretext
xsect_pretext_option	integer	X-Section pretext option, 0: None, 1: Automatic, 2: Manual
xsectionFirst (read only)	integer	First cross section ID
xsectionLast (read only)	integer	Last cross section ID

Detailed Description

The Belt class allows you to create, modify, and manipulate belt fitting definitions. See the documentation below for more details.

Constructor

new Belt(model*[Model]*, id*[integer]*, title (optional)*[string]*, structural_type (optional)*[string]*, flag (optional)*[integer]*)

Description

Create a new Belt object.

Arguments

- **model** (Model)

Model that the belt definition will be created in

- **id** (integer)

Belt number

- **title (optional)** (string)

Title for the belt

- **structural_type (optional)** (string)

Seatbelt will be fitted around this entity type. This will trigger creation of sets as required. Type can be one of MODEL, DUMMY, PART, any ELEMENT subtype such as SHELL, or any SET subtype such as SET_PART. See Appendix I of the PRIMER manual for more information on PRIMER types

- **flag (optional)** (integer)

Flag used to identify entities that the belt should fit around. This argument is ignored if structural_type is MODEL. Instead, the current model is used

Returns

Belt object

Return type

Belt

Example

To create a new belt called 'Example' in model m with label 100:

```
var b = new Belt(m, 100, 'Example');
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a belt.

Arguments

- **Comment** (Comment)

Comment that will be attached to the belt

Returns

No return value

Example

To associate comment c to the belt b:

```
b.AssociateComment(c);
```

Blank()

Description

Blanks the belt

Arguments

No arguments

Returns

No return value

Example

To blank belt b:

```
b.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the belts in the model.

Arguments

- **Model** (Model)

Model that all belts will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the belts in model m:

```
Belt.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged belts in the model.

Arguments

- **Model** (Model)

Model that all the flagged belts will be blanked in

- **flag** (Flag)

Flag set on the belts that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the belts in model m flagged with f:

```
Belt.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the belt is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if belt b is blanked:

```
if (b.Blanked() ) do_something...
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the belt.

Arguments

- **flag** (Flag)

Flag to clear on the belt

Returns

No return value

Example

To clear flag f for belt b:

```
b.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the belt. The target include of the copied belt can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

Belt object

Return type

Belt

Example

To copy belt b into belt z:

```
var z = b.Copy();
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a belt.

Arguments

- **Comment** (Comment)

Comment that will be detached from the belt

Returns

No return value

Example

To detach comment c from the belt b:

```
b.DetachComment(c);
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for belt with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The belt object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for belt b:

```
b.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for belt b:

```
b.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for belt. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for belt b:

```
b.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first belt in the model.

Arguments

- **Model** (Model)

Model to get first belt in

Returns

Belt object (or null if there are no belts in the model).

Return type

Belt

Example

To get the first belt in model m:

```
var b = Belt.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free belt label in the model. Also see Belt.LastFreeLabel(), Belt.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free belt label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

Belt label.

Return type

Number

Example

To get the first free belt label in model m:

```
var label = Belt.FirstFreeLabel(m);
```

Fit()

Description

(Re)fits belt

Arguments

No arguments

Returns

No return value

Example

To (re)fit belt b:

```
b.Fit();
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the belts in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all belts will be flagged in

- **flag** (Flag)

Flag to set on the belts

Returns

No return value

Example

To flag all of the belts with flag f in model m:

```
Belt.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the belt is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the belt

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if belt b has flag f set on it:

```
if (b.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each belt in the model.
Note that ForEach has been designed to make looping over belts as fast as possible and so has some limitations. Firstly, a single temporary Belt object is created and on each function call it is updated with the current belt data. This means that you should not try to store the Belt object for later use (e.g. in an array) as it is temporary. Secondly, you cannot create new belts inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all belts are in

- **func** (function)

Function to call for each belt

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the belts in model m:

```
Belt.ForEach(m, test);
function test(b)
{
// b is Belt object
}
```

To call function test for all of the belts in model m with optional object:

```
var data = { x:0, y:0 };
Belt.ForEach(m, test, data);
function test(b, extra)
{
// b is Belt object
// extra is data
}
```

Generate()

Description

Generates belt mesh. Extracts and uses existing mesh properties when a mesh is present; inserts a default mesh otherwise.

Arguments

No arguments

Returns

No return value

Example

To generate a mesh for belt b:

```
b.Generate();
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Belt objects or properties for all of the belts in a model in PRIMER. If the optional property argument is not given then an array of Belt objects is returned. If the property argument is given, that property value for each belt is returned in the array instead of a Belt object

Arguments

- **Model** (Model)

Model to get belts from

- **property (optional)** (string)

Name for property to get for all belts in the model

Returns

Array of Belt objects or properties

Return type

Array

Example

To make an array of Belt objects for all of the belts in model m:

```
var a = Belt.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each belt in model m:

```
var a = Belt.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a belt.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the belt b:

```
var comm_array = b.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of Belt objects for all of the flagged belts in a model in PRIMER If the optional property argument is not given then an array of Belt objects is returned. If the property argument is given, then that property value for each belt is returned in the array instead of a Belt object

Arguments

- **Model** (Model)

Model to get belts from

- **flag** (Flag)

Flag set on the belts that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged belts in the model

Returns

Array of Belt objects or properties

Return type

Array

Example

To make an array of Belt objects for all of the belts in model m flagged with f:

```
var b = Belt.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the belts in model m flagged with f:

```
var a = Belt.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the Belt object for a belt ID.

Arguments

- **Model** (Model)

Model to find the belt in

- **number** (integer)

number of the belt you want the Belt object for

Returns

Belt object (or null if belt does not exist).

Return type

Belt

Example

To get the Belt object for belt 100 in model m

```
var b = Belt.GetFromID(m, 100);
```

GetMesh(index*[integer]*)

Description

Returns the information for a belt mesh section (properties base_pt1, base_pt2, path_pt1, path_pt2, mode, lb1, lb2). See Belt.SetMesh() for more information on supported properties. Must be preceded by a call to Belt.Generate().

Arguments

- **index** (integer)

The index of the mesh section you want the information for. **Note that mesh segments start at 0, not 1**. 0 <= index < meshSegs

Returns

Object containing the mesh section information

Return type

Object

Example

To get the information for the 3rd mesh section for belt b:

```
var info = b.GetMesh(2);
```

GetParameter(prop*[string]*)

Description

Checks if a Belt property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the Belt.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

belt property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if Belt property b.example is a parameter:

```
Options.property_parameter_names = true;
if (b.GetParameter(b.example) ) do_something...
Options.property_parameter_names = false;
```

To check if Belt property b.example is a parameter by using the GetParameter method:

```
if (b.ViewParameters().GetParameter(b.example) ) do_something...
```

GetPoint(index*[integer]*)

Description

Returns the information for a path point (properties fixity, x, y, z, node, trx1, try1, trz1, tnx1, tny1, tnz1, tnode1, trx2, try2, trz2, tnx2, tny2, tnz2, tnode2). Properties fixity, x, y, z and node will always be returned. Twist properties trx1, try1, trz1, tnx1, tny1, tnz1, tnode1, trx2, try2, trz2, tnx2, tny2, tnz2 and tnode2 will only be returned if defined for the point.

Arguments

- **index** (integer)

The index of the path point you want the information for. **Note that path points start at 0, not 1**. 0 <= index < points

Returns

Object containing the path point information

Return type

Object

Example

To get the information for the 3rd path point for belt b:

```
var info = b.GetPoint(2);
```

InsertPoint(index*[integer]*, position*[integer]*, data*[object]*)

Description

Inserts a path point before/after an existing one. Subsequent path points will be moved 'up' as required.

Arguments

- **index** (integer)

The index of an existing path point. **Note that path points start at 0, not 1**. 0 <= index < points

- **position** (integer)

Do we want to insert before or after the path point denoted by index? The position can be Belt.INSERT_AFTER or Belt.INSERT_BEFORE

- **data** (object)

Object containing the path point data.

Object has the following properties:

Name	Type	Description
fixity	integer	Point fixity type. Bitwise 'or' of the Path point fixity constants: Belt.B_POST_SLIPRING, Belt.FREE_SLIPRING, Belt.KNOWN, Belt.RETRACTOR, Belt.TWIST, Belt.XSEC

node (optional)	integer	Node label (not required if using x, y and z)
tnode1 (optional)	integer	Twist node 1 label
tnode2 (optional)	integer	Twist node 2 label
tnx1 (optional)	real	X component of normal vector 1
tnx2 (optional)	real	X component of normal vector 2
tny1 (optional)	real	Y component of normal vector 1
tny2 (optional)	real	Y component of normal vector 2
tnz1 (optional)	real	Z component of normal vector 1
tnz2 (optional)	real	Z component of normal vector 2
trx1 (optional)	real	X component of twist radial vector 1
trx2 (optional)	real	X component of twist radial vector 2
try1 (optional)	real	Y component of twist radial vector 1
try2 (optional)	real	Y component of twist radial vector 2
trz1 (optional)	real	Z component of twist radial vector 1
trz2 (optional)	real	Z component of twist radial vector 2
x (optional)	real	X coordinate (not required if using node)
y (optional)	real	Y coordinate (not required if using node)
z (optional)	real	Z coordinate (not required if using node)

Returns

no return value

Example

To insert a new 2nd path point for belt b with B-Post slipring fixity using twist nodes 1000 and 10001:

```
var data = { fixity:Belt.RETRACTOR|Belt.TWIST, node:999 tnode1:1000, tnode2:1001
};
b.InsertPoint(1, Belt.INSERT_BEFORE, data);
```

Last(Model[*Model*]) [static]

Description

Returns the last belt in the model.

Arguments

- **Model** (Model)

Model to get last belt in

Returns

Belt object (or null if there are no belts in the model).

Return type

Belt

Example

To get the last belt in model m:

```
var b = Belt.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free belt label in the model. Also see Belt.FirstFreeLabel(), Belt.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free belt label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

Belt label.

Return type

Number

Example

To get the last free belt label in model m:

```
var label = Belt.LastFreeLabel(m);
```

Next()

Description

Returns the next belt in the model.

Arguments

No arguments

Returns

Belt object (or null if there are no more belts in the model).

Return type

Belt

Example

To get the belt in model m after belt b:

```
var b = b.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) belt label in the model. Also see Belt.FirstFreeLabel(), Belt.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free belt label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

Belt label.

Return type

Number

Example

To get the next free belt label in model m:

```
var label = Belt.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a belt.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only belts from that model can be picked. If the argument is a Flag then only belts that are flagged with *limit* can be selected. If omitted, or null, any belts from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

Belt object (or null if not picked)

Return type

Belt

Example

To pick a belt from model m giving the prompt 'Pick belt from screen':

```
var b = Belt.Pick('Pick belt from screen', m);
```

Previous()

Description

Returns the previous belt in the model.

Arguments

No arguments

Returns

Belt object (or null if there are no more belts in the model).

Return type

Belt

Example

To get the belt in model m before belt b:

```
var b = b.Previous();
```

RemovePoint(index*[integer]*)

Description

Removes a path point from a belt

Arguments

- **index** (integer)

The index of the path point you want to remove. **Note that path points start at 0, not 1**. 0 <= index < points

Returns

no return value

Example

To remove for the 3rd path point for belt b:

```
b.RemovePoint(2);
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the belts in the model.

Arguments

- **Model** (Model)

Model that all belts will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the belts in model m, from 1000000:

```
Belt.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged belts in the model.

Arguments

- **Model** (Model)

Model that all the flagged belts will be renumbered in

- **flag** (Flag)

Flag set on the belts that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the belts in model m flagged with f, from 1000000:

```
Belt.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select belts using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting belts

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only belts from that model can be selected. If the argument is a Flag then only belts that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any belts can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of belts selected or null if menu cancelled

Return type

Number

Example

To select belts from model m, flagging those selected with flag f, giving the prompt 'Select belts':

```
Belt.Select(f, 'Select belts', m);
```

To select belts, flagging those selected with flag f but limiting selection to belts flagged with flag l, giving the prompt 'Select belts':

```
Belt.Select(f, 'Select belts', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the belt.

Arguments

- **flag** (Flag)

Flag to set on the belt

Returns

No return value

Example

To set flag f for belt b:

```
b.SetFlag(f);
```

SetMesh(index*[integer]*, data*[object]*)

Description

Sets the data for various properties for a mesh section in a belt. Values for properties not invoked will be retained as is. Must be preceded by a call to Belt.Generate()

Arguments

- **index** (integer)

The index of the mesh section you want to set. **Note that mesh segments start at 0, not 1**

- **data** (object)

Object containing the mesh section data.

Object has the following properties:

Name	Type	Description
base_pt1	integer	1st base point number
base_pt2	integer	2nd base point number

lb1	integer	Number of belt elements at the 1st end for mixed modes
lb2	integer	Number of belt elements at the 2nd end for mixed modes
mode	integer	Meshing modes can be of old style or new style. The following old style constants are available: Belt.MSEG_B1_ONLY, Belt.MSEG_B2_ONLY, Belt.MSEG_SH_ONLY, Belt.MSEG_MIX_SB1, Belt.MSEG_MIX_SB2 The following constant must be invoked in order to use the new style: Belt.MSEG_BD_NEW The following new style constants are available: Belt.MSEG_E1_1D, Belt.MSEG_E1_2D, Belt.MSEG_E1_SH, Belt.MSEG_E2_1D, Belt.MSEG_E2_2D, Belt.MSEG_E2_SH, Belt.MSEG_CE_1D, Belt.MSEG_CE_2D, Belt.MSEG_CE_SH
path_pt1	integer	1st path point number
path_pt2	integer	2nd path point number
retractor (optional)	integer	Retractor id to be used

Returns

no return value

Example

To set the following properties for the final mesh section: base points: 5, 9, path points: 59, 92, mode: 1D at ends, shells at centre, number of elements at either end: 4 and 10:

```
var data = { base_pt1: 5, base_pt2: 9, path_pt1: 59, path_pt2: 92, mode:
Belt.MSEG_BD_NEW | Belt.MSEG_E1_1D | Belt.MSEG_CE_SH | Belt.MSEG_E2_1D, lb1: 4,
lb2: 10 };
b.SetMesh(b.meshSegs-1, data);
```

SetMeshingLabels(entity_type*[constant]*, label_value*[integer]*) [static]

Description

Set the start labels for the entities created for a Seat Belt.

Arguments

- **entity_type** (constant)

The Meshing label can be Belt.MESH_NODE, Belt.MESH_SHELL, Belt.MESH_SET_NODE, Belt.MESH_SET_SHELL, Belt.MESH_SEATBELT, Belt.MESH_NRBC, BELT.MESH_RETRACTOR, Belt.MESH_XSEC, Belt.MESH_SLIPRING, Belt.MESH_SET_PART, Belt.MESH_2D_SLIPRING_SET_NODE, Belt.MESH_ALL.

- **label_value** (integer)

The initial label value to be assigned for the entity type.

Returns

no return value

Example

To get the initial value of the node label in seatbelt meshing as 1000:

```
Belt.SetMeshingLabels(Belt.MESH_NODE,1000)
```

SetPoint(index*[integer]*, data*[object]*)

Description

Sets the data for a path point in a belt

Arguments

- **index** (integer)

The index of the path point you want to set. **Note that path points start at 0, not 1**. To add a new point use index points

- **data** (object)

Object containing the path point data.

Object has the following properties:

Name	Type	Description
fixity	integer	Point fixity type. Bitwise 'or' of the Path point fixity constants: Belt.B_POST_SLIPRING, Belt.FREE_SLIPRING, Belt.KNOWN, Belt.RETRACTOR, Belt.TWIST, Belt.XSEC
node (optional)	integer	Node label (not required if using x, y and z)
tnode1 (optional)	integer	Twist node 1 label
tnode2 (optional)	integer	Twist node 2 label
tnx1 (optional)	real	X component of normal vector 1
tnx2 (optional)	real	X component of normal vector 2
tny1 (optional)	real	Y component of normal vector 1
tny2 (optional)	real	Y component of normal vector 2
tnz1 (optional)	real	Z component of normal vector 1
tnz2 (optional)	real	Z component of normal vector 2
trx1 (optional)	real	X component of twist radial vector 1
trx2 (optional)	real	X component of twist radial vector 2
try1 (optional)	real	Y component of twist radial vector 1
try2 (optional)	real	Y component of twist radial vector 2
trz1 (optional)	real	Z component of twist radial vector 1
trz2 (optional)	real	Z component of twist radial vector 2
x (optional)	real	X coordinate (not required if using node)
y (optional)	real	Y coordinate (not required if using node)
z (optional)	real	Z coordinate (not required if using node)

Returns

no return value

Example

To add a new B-Post slipring path point to belt b at node 1000:

```
var data = { fixity:Belt.B_POST_SLIPRING, node:1000 };
b.SetPoint(b.points, data);
```

To add a new path point to belt b at coordinate (10, 20, 30):

```
var data = { fixity:0, x:10, y:20, z:30 };
b.SetPoint(b.points, data);
```

To add a new retractor path point to belt b at (10, 20, 30) with twist nodes 1000 and 1001:

```
var data = { fixity:Belt.RETRACTOR|Belt.TWIST, x:10, y:20, z:30, tnode1:1000,
tnode2:1001 };
b.SetPoint(b.points, data);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the belt. The belt will be sketched until you either call Belt.Unsketch(), Belt.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the belt is sketched. If omitted redraw is true. If you want to sketch several belts and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch belt b:

```
b.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged belts in the model. The belts will be sketched until you either call Belt.Unsketch(), Belt.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged belts will be sketched in

- **flag** (Flag)

Flag set on the belts that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the belts are sketched. If omitted redraw is true. If you want to sketch flagged belts several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all belts flagged with flag in model m:

```
Belt.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of belts in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing belts should be counted. If false or omitted referenced but undefined belts will also be included in the total.

Returns

number of belts

Return type

Number

Example

To get the total number of belts in model m:

```
var total = Belt.Total(m);
```

Unblank()

Description

Unblanks the belt

Arguments

No arguments

Returns

No return value

Example

To unblank belt b:

```
b.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the belts in the model.

Arguments

- **Model** (Model)

Model that all belts will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the belts in model m:

```
Belt.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged belts in the model.

Arguments

- **Model** (Model)

Model that the flagged belts will be unblanked in

- **flag** (Flag)

Flag set on the belts that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the belts in model m flagged with f:

```
Belt.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the belts in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all belts will be unset in

- **flag** (Flag)

Flag to unset on the belts

Returns

No return value

Example

To unset the flag f on all the belts in model m:

```
Belt.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the belt.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the belt is unsketched. If omitted redraw is true. If you want to unsketch several belts and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch belt b:

```
b.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all belts.

Arguments

- **Model** (Model)

Model that all belts will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the belts are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all belts in model m:

```
Belt.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged belts in the model.

Arguments

- **Model** (Model)

Model that all belts will be unsketched in

- **flag** (Flag)

Flag set on the belts that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the belts are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all belts flagged with flag in model m:

```
Belt.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

Belt object.

Return type

Belt

Example

To check if Belt property b.example is a parameter by using the Belt.GetParameter() method:

```
if (b.ViewParameters().GetParameter(b.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for belt with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description

autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The belt object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for belt b:

```
b.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for belt b:

```
b.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for belt. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for belt b:

```
b.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this belt.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for belt b:

```
var xrefs = b.Xrefs();
```

Check class

The Check class enables you to access model checking in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- AddDashboardComment(comment*[string]*)
- AddDashboardHealth(model health*[String]*, Health text colour (optional)*[constant]*, Health button colour(optional)*[constant]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- GetAllDashboards()
- KeyoutHook(interrupt flag*[boolean]*)
- OnDismiss(name*[function]*)
- OnRecheck(name*[function]*)
- SetDashboardMessage(first*[string]*, second (optional)*[string]*)
- SetDashboardStatus(status*[constant]*)
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]

Check constants

Constants for Dashboard

Name	Description
Check.ERROR	Dashboard check gave error(s)
Check.OK	Dashboard check status OK
Check.UNKNOWN	Dashboard check status unknown (not run)
Check.WARNING	Dashboard check gave warning(s)

Constants for dashboard health colour

Name	Description
Check.BLACK	Colour black
Check.BLUE	Colour blue
Check.CYAN	Colour cyan
Check.DARKBLUE	Colour dark blue
Check.DARKGREEN	Colour dark green
Check.DARKGREY	Colour dark grey
Check.DARKRED	Colour dark red
Check.GREEN	Colour green
Check.GREY	Colour grey
Check.LIGHTGREY	Colour light grey

Check.MAGENTA	Colour magenta
Check.ORANGE	Colour orange
Check.RED	Colour red
Check.WHITE	Colour white
Check.YELLOW	Colour yellow

Detailed Description

The Check class is used add checks to PRIMER using JavaScript. Two different types of checks can be added:

- Individual checks for each node, part, shell etc in a model.
- Custom checks that can reference multiple entities for checking in a model

By default PRIMER will look in 4 locations for additional JavaScript checks to run when doing checking:

- OA_ADMIN/primer_library/scripts/checks
- OA_INSTALL/primer_library/scripts/checks
- HOME/primer_library/scripts/checks
- <current directory>/primer_library/scripts/checks

The directories OA_INSTALL/primer_library/scripts etc can be changed with the primer*script_dir preference. In addition, PRIMER will also use a custom preference directory specified with the -pref command line option.

For individual checks PRIMER will look in these directories for a script with the name 'class_name.js'. For example if you wanted to write a script that will be run for every part in a model the script should be called 'Part.js'.
For custom checks PRIMER will look in these directories for a script called 'custom.js'. This obviously means that there can only be one custom script in each directory. **These filenames are case sensitive**.
Individual scripts will be called with 3 arguments:
arguments[0] = Name of the script
arguments[1] = model object
arguments[2] = Item object

Individual scripts can add warnings or errors by using the Warning() or Error() methods of the appropriate class. For example for a Part the script can call the methods Part.Error() and Part.Warning().
The script should not call the Error() and Warning() methods of other classes.

As models in PRIMER can easily contain millions of items (e.g. nodes or shells), PRIMER runs checking scripts in a special way. The script is not compiled, run and destroyed for every instance of the item that is checked as otherwise this would be incredibly slow. Instead the script is only compiled **once** and is **executed** multiple times, once for every instance of the item in the model. Due to this, the checking script must be written carefully and some JavaScript statements (e.g. global variables declared with let) will not work. The examples below are designed to be simple to illustrate how the methods work. Please see the PRIMER manual section 'User Defined Checks Using JavaScript' in 'MODEL > CHECK' for more details.

As a simple example of an individual check, suppose you wanted it to be an error if any shell parts in your model did not use type 16 shells. Add a script called 'Part.js' in the directory 'OA_INSTALL/primer_library/scripts/checks' (or one of the other directories) containing:

```
// arguments[0] is name of script
var m = arguments[1]; // arguments[1] is model pointer
var p = arguments[2]; // arguments[2] is part pointer
if (p.exists && p.secid)
{
    var s = Section.GetFromID(m, p.secid);
    if (s.exists && s.type == Section.SHELL && s.elform != 16)
        p.Error("Shell part elform not 16", { details: "Fictional company policy
is to use elform 16 for shell parts" } );
}
```

Autofixes can be defined for checks by providing a function and using the autofixFunction and autofixMessage properties in the Error or Warning methods of the appropriate class. For example, the simple part check above could also be defined with an autofix to change the element formulation on the section card to be type 16.

```
var m = arguments[1]; // arguments[1] is model pointer
var p = arguments[2]; // arguments[2] is part pointer
if (p.exists && p.secid)
{
    var s = Section.GetFromID(m, p.secid);
    if (s.exists && s.type == Section.SHELL && s.elform != 16)
        p.Error("Shell part elform not 16", { autofixFunction: updateElform,
autofixMessage: "Update element formulation to 16" );
}
```

```
function updateElform()
{
    var p = this;
    if (p.exists && p.secid)
    {
        var s = Section.GetFromID(m, p.secid);
        if (s.exists && s.type == Section.SHELL && s.elform != 16)
        {
            s.elform = 16;
            return 'Updated section ${s.label} to elform 16';
        }
    }
    return 'Error updating element formulation for part ${p.label}';
}
```

Custom scripts will be called with 2 arguments:
arguments[0] = Name of the script
arguments[1] = model object
Custom scripts can add warnings or errors by using the static Check.Error() and Check.Warning() methods.
The script should not call the Error() and Warning() methods of other classes.

As a simple example of a custom check, suppose a dummy uses node 1000 for the H-point and this should be at coordinates (1000, -500, 100) within tolerance of 0.1 for an analysis . You do not want to run a check for every node in the model (i.e. an individual check). You just want to check that node 1000 is at the correct coordinates. To do this you could create a script called 'custom.js' in the directory 'OA_INSTALL/primer_library/scripts/checks' (or one of the other directories) containing:

```
// arguments[0] is name of script
var m = arguments[1]; // arguments[1] is model pointer
var n = Node.GetFromID(m, 1000);
if (!n)
    Check.Error("No H-point node", { details: "Model does not contain node for
dummy H-point" } );
if (!n.exists)
    Check.Error("H-point node not defined", { details: "Dummy H-point node is
referred to but not defined" } );
var dx = n.x - 1000;
var dy = n.y - (-500);
var dy = n.z - 100;
var d = Math.sqrt(dx*dx + dy*dy + dz*dz);
if (d > 0.1)
    Check.Error("H-point not at correct position", { details: 'Dummy H-point is
${d}mm away from target position' } );
```

See the documentation below for more details.

Details of functions

AddDashboardComment(comment*[string]*) [static]

Description

Adds a comment for a user dashboard check. Multiple comments can be added. Call this function as many times as required.
This function should only be called from a user JavaScript dashboard script.

Arguments

- **comment** (string)

The comment to add.

Returns

No return value

Example

To add a comment:

```
Check.AddDashboardComment("This is a comment");
```

AddDashboardHealth(model health*[String]*, Health text colour (optional)*[constant]*, Health button colour(optional)*[constant]*) [static]

Description

Allows the user to add the value of model health based on the other dashboard results
This function should only be called from model_health.config.js which should be placed with the other user defined dashboard scripts.

Arguments

- **model health** (String)

Text which will be displayed on the dashboard panel and the summary files.

- **Health text colour (optional)** (constant)

Colour of the model health text. The default colour is Black.

- **Health button colour(optional)** (constant)

Colour of the model health button. The default colour is dark grey.

Returns

No return value

Example

To add computed health as "Model Health 85.1%" and the text colour to red and the button colour to green

```
Check.AddDashboardHealth("Model Health 85.1%",Check.RED,Check.GREEN);
```

Error(message*[string]*, options (optional)*[object]*) [static]

Description

Adds a custom error with an optional custom autofix. This function should only be called from a custom JavaScript check script. See the details in the Check class for how to do this.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply a custom autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The model object for the model being checked is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...

autofixMessage (optional)	string	A message to describe a custom autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error":

```
Check.Error('My custom error');
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix:

```
Check.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [static] [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a custom error. This function should only be called from a custom JavaScript check script. See the details in the Check class for how to do this.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error":

```
Check.Error('My custom error');
```

GetAllDashboards() [static]

Description

Returns data from all the dashboards that are defined.
This function should only be called from model_health.config.js which should be placed with the other user defined dashboard scripts.
The dashboard properties are:

- result (Overall result of the dashboard)
- title (dashboard title)
- message1 (First message of the dashboard)
- message2 (Second message of the dashboard)
- comments (Array of comments on the dashboard)

Arguments

No arguments

Returns

Array of objects with the following properties:

Name	Type	Description
comments	Array of string	Comments on the dashboard
message1	string	First message of the dashboard
message2	string	Second message of the dashboard
result	integer	Overall result of the dashboard.
title	string	Dashboard title

Return type

object

Example

To get the status of all the dashboards:

```
Check.GetAllDashboards();
```

For more details on how to use this function, please take a look at the example script model_health.config.js which is present in the dashboard scrips folder

---

KeyoutHook(interrupt flag*[boolean]*) [static]

Description

Used to proceed with or abort the keyout operation (Ansys LS-DYNA output) from the keyout_hook.js script. The current hooks are launched just before the keyout operation from the model write tab, writing from the dialogue box and during keyout from the include tree. Please look at the example_keyout_script.js for an example of its usage.

Arguments

- **interrupt flag** (boolean)

If this flag is set to true then keyout is aborted else keyout proceeds as usual.

Returns

No return value

Example

To abort a keyout, set the following line in keyout_hook.js:

```
Check.KeyoutHook(true);
```

OnDismiss(name*[function]*) [static]

Description

Set a function to call when model checking is dismissed. This function will be called when the user presses dismiss in the model checking panel. This allows you to add a function to do something when checking finishes. In normal usage it is very unlikely that you will need to use this function. However it is provided in the rare case that it could be useful. Please see the PRIMER manual section 'User Defined Checks Using JavaScript' in 'MODEL > CHECK' for more details.
The function should not have any arguments defined.
To remove the function use Check.OnDismiss(null).
This method should only be called from a JavaScript check script.

Arguments

- **name** (function)

The name of the function (or null to remove a function)

Returns

No return value

Example

To set function MyDismissFunction as the function to call when model checking is dismissed:

```
Check.OnDismiss(MyDismissFunction);
```

OnRecheck(name*[function]*) [static]

Description

Set a function to call when model checking is rechecked for this type. This function will be called once before any checks are rerun for this type. This allows you to add a function to do something when checking restarts for a type. In normal usage it is very unlikely that you will need to use this function. However it is provided in the rare case that it could be useful. Please see the PRIMER manual section 'User Defined Checks Using JavaScript' in 'MODEL > CHECK' for more details.
The function should not have any arguments defined.
To remove the function use Check.OnRecheck(null).
This method should only be called from a JavaScript check script.

Arguments

- **name** (function)

The name of the function (or null to remove a function)

Returns

No return value

Example

To set function MyRecheckFunction as the function to call when this type is rechecked:

```
Check.OnRecheck(MyRecheckFunction);
```

SetDashboardMessage(first*[string]*, second (optional)*[string]*) [static]

Description

Adds a message for a user dashboard check. Each dashboard can currently show two messages.
This function should only be called from a user JavaScript dashboard script.

Arguments

- **first** (string)

The first message to add.

- **second (optional)** (string)

The second message to add.

Returns

No return value

Example

To add the message with two lines:

```
Check.SetDashboardMessage("This is a message", "shown on two lines");
```

To add the message with one line:

```
Check.SetDashboardMessage("This is a single message");
```

SetDashboardStatus(status*[constant]*) [static]

Description

Sets the status of a user dashboard check.
This function should only be called from a user JavaScript dashboard script.

Arguments

- **status** (constant)

The status. Can be Check.OK, Check.WARNING, Check.ERROR or Check.UNKNOWN.

Returns

No return value

Example

To set the status to OK (green):

```
Check.SetDashboardStatus(Check.OK);
```

Warning(message*[string]*, options (optional)*[object]*) [static]

Description

Adds a custom warning with an optional custom autofix. This function should only be called from a custom JavaScript check script. See the details in the Check class for how to do this.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply a custom autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The model object for the model being checked is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe a custom autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning":

```
Check.Warning('My custom warning');
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix:

```
Check.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [static] [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a custom warning. This function should only be called from a custom JavaScript check script. See the details in the Check class for how to do this.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning":

```
Check.Warning('My custom warning');
```

Colour class

The Colour class contains constants relating to colours. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- GetFromName(name*[string]*)
- RGB(red*[integer]*, green*[integer]*, blue*[integer]*)

Colour constants

Name	Description
Colour.ASSEMBLY	Base colour on assembly
Colour.BACKGROUND	Background colour
Colour.BLACK	Colour black
Colour.BLUE	Colour blue
Colour.CYAN	Colour cyan
Colour.DARK_ORANGE	Colour dark orange
Colour.DEFAULT	Default colour for objects
Colour.GREEN	Colour green
Colour.GREEN_CYAN	Colour green/cyan
Colour.GREY	Colour grey
Colour.INCLUDE	Base colour on include file
Colour.LIGHT_BLUE	Colour light blue
Colour.MAGENTA	Colour magenta
Colour.MATERIAL	For elements with part IDs base colour on material ID
Colour.MEDIUM_BLUE	Colour medium blue
Colour.MODEL	Base colour on model
Colour.NOT_BACKGROUND	Not the background colour
Colour.ORANGE	Colour orange
Colour.PART	For elements with part IDs base colour on part ID
Colour.RED	Colour red
Colour.RED_MAGENTA	Colour red/magenta
Colour.SECTION	For elements with part IDs base colour on section ID
Colour.SKETCH	Sketch colour
Colour.TEXT	Text colour

Colour.WHITE	Colour white
Colour.YELLOW	Colour yellow
Colour.YELLOW_GREEN	Colour yellow/green

Detailed Description

The Colour class is used to define colours, either by predefined colours or by RGB values. The easiest way to set the colour of something is to use the predefined colour constants. e.g. to set the colour of part p to red:

```
p.colour = Colour.RED;
```

For other colours use Colour.RGB().

Details of functions

GetFromName(name*[string]*) [static]

Description

Returns the colour for a given core or user colour name

Arguments

- **name** (string)

The name of the colour, for example red or user_green or green/cyan.

Returns

colour value (integer)

Return type

Number

RGB(red*[integer]*, green*[integer]*, blue*[integer]*) [static]

Description

Creates a colour from red, green and blue components

Arguments

- **red** (integer)

red component of colour (0-255).

- **green** (integer)

green component of colour (0-255).

- **blue** (integer)

blue component of colour (0-255).

Returns

colour value (integer)

Return type

Number

Example

To set the colour of model m to red:

```
m.SetColour( Colour.RGB(255, 0, 0) );
```

To set the colour of part p to red:

```
p.colour = Colour.RGB(255, 0, 0);
```

Conx class

The Conx class gives you access to connections in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model *[Model]*, redraw (optional) *[boolean]*)
- BlankFlagged(Model *[Model]*, flag *[Flag]*, redraw (optional) *[boolean]*)
- First(Model *[Model]*)
- FirstFreeLabel(Model *[Model]*, layer (optional) *[Include number]*)
- FlagAll(Model *[Model]*, flag *[Flag]*)
- ForEach(Model *[Model]*, func *[function]*, extra (optional) *[any]*)
- GetAll(Model *[Model]*, property (optional) *[string]*)
- GetFlagged(Model *[Model]*, flag *[Flag]*, property (optional) *[string]*)
- GetFromID(Model *[Model]*, number *[integer]*)
- Last(Model *[Model]*)
- LastFreeLabel(Model *[Model]*, layer (optional) *[Include number]*)
- NextFreeLabel(Model *[Model]*, layer (optional) *[Include number]*)
- Pick(prompt *[string]*, limit (optional) *[Model or Flag]*, modal (optional) *[boolean]*, button text (optional) *[string]*)
- RealizeAll(Model *[Model]*)
- RealizeFlagged(Model *[Model]*, flag *[Flag]*)
- ReloadConnectors()
- RenumberAll(Model *[Model]*, start *[integer]*)
- RenumberFlagged(Model *[Model]*, flag *[Flag]*, start *[integer]*)
- Select(flag *[Flag]*, prompt *[string]*, limit (optional) *[Model or Flag]*, modal (optional) *[boolean]*)
- SetRuleDiameter(diameter *[integer]*)
- SetRuleFEPID(pid *[integer]*)
- SetRulePID(pid *[integer]*)
- SketchFlagged(Model *[Model]*, flag *[Flag]*, redraw (optional) *[boolean]*)
- Total(Model *[Model]*, exists (optional) *[boolean]*)
- UnblankAll(Model *[Model]*, redraw (optional) *[boolean]*)
- UnblankFlagged(Model *[Model]*, flag *[Flag]*, redraw (optional) *[boolean]*)
- UnflagAll(Model *[Model]*, flag *[Flag]*)
- UnsketchAll(Model *[Model]*, redraw (optional) *[boolean]*)
- UnsketchFlagged(Model *[Model]*, flag *[Flag]*, redraw (optional) *[boolean]*)
- UseParentLayer(option *[boolean]*)
- UseSPR2Pref(option *[boolean]*)

Member functions

- AssociateComment(Comment *[Comment]*)
- Blank()
- Blanked()
- ClearFlag(flag *[Flag]*)
- Copy(range (optional) *[boolean]*)
- DetachComment(Comment *[Comment]*)
- EmptyPatch()
- Error(message *[string]*, options (optional) *[object]*)
- Error(message *[string]*, details (optional) *[string]*) [deprecated]
- ExtractColour()
- Flagged(flag *[Flag]*)
- GetComments()
- GetElements()
- GetEntities(type *[string]*)
- GetLayerData(layer *[integer]*)
- GetLayerShells(layer *[integer]*)
- GetParameter(prop *[string]*)

- GetPatchCoords(point*[integer]*)
- GetPatchTopol(point*[integer]*)
- GetPathData(point*[integer]*)
- GetPidData()
- GetSettings()
- GetShellThickness(Layer*[integer]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- RemovePatchTopol(layer*[integer]*)
- RemovePathData(layer*[integer]*)
- SetFlag(flag*[Flag]*)
- SetLayerData(layer*[integer]*, items*[integer or string]*)
- SetPatchCoords(point*[integer]*, x*[real]*, y*[real]*, z*[real]*)
- SetPatchTopol(point*[integer]*, c1*[integer]*, c2*[integer]*, c3*[integer]*, c4 (optional)*[integer]*)
- SetPathData(point*[integer]*, x*[real]*, y*[real]*, z*[real]*)
- SetPidData(items*[integer or string]*)
- SetSettings(data*[object]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

Conx constants

Name	Description
Conx.ASSEMBLY	If the connection refers to an assembly rather than individual layers, the assembly is defined by part tree assembly.
Conx.PART_SET	If the connection refers to an assembly rather than individual layers, the assembly is defined by part set.

Constants for adhesive connections

Name	Description
Conx.ADHESIVE_PATCH	Connection adhesive type is a patch.
Conx.ADHESIVE_SOLID	Connection adhesive type is a solid line.

Constants for bolt connections

Name	Description
Conx.BOLT_MODULE	Library bolt.
Conx.BOLT_MRG_2PTS	2pt Patch Beam.
Conx.BOLT_MRG_2PTS_RB	2pt Patch (Rigid Beam).
Conx.BOLT_MRG_2PTS_RJ	2pt Patch Revolute joint.
Conx.BOLT_MRG_CYL	Cylindrical Merge.
Conx.BOLT_MRG_CYL_BALL	Cylindrical Patch Ball joint.
Conx.BOLT_MRG_CYL_BEAM	Cylindrical Patch Beam.
Conx.BOLT_NRB_2PTS	2pt NRB Beam.
Conx.BOLT_NRB_CYL	Cylindrical NRB.
Conx.BOLT_NRB_CYL_BALL	Cylindrical NRB Ball joint.

Conx.BOLT_NRB_CYL_BEAM	Cylindrical NRB Beam.
Conx.BOLT_NRB_SPH	Spherical NRB.
Conx.BOLT_NRB_SPH_BALL	Spherical NRB Ball joint.
Conx.BOLT_NRB_SPH_DISC	Spherical NRB Discrete Beam.

Constants for connection status

Name	Description
Conx.BAD	Connection is bad (e.g. necessary data is missing).
Conx.DORMANT	Connection is dormant (not yet made).
Conx.INVALID	Connection has been made but something is wrong (e.g. part moved).
Conx.MADE	Connection has been made but status is unknown.
Conx.REALIZED	Connection has been made and is OK (checks OK).

Constants for connection type

Name	Description
Conx.ADHESIVE	Connection is adhesive.
Conx.BOLT	Connection is a bolt.
Conx.RIVET	Connection is rivet.
Conx.SPOTWELD	Connection is a spotweld.
Conx.SPOTWELD_LINE	Connection is a spotweld line.

Constants for spotweld connections

Name	Description
Conx.SPOTWELD_BEAM	Connection spotweld type is beam.
Conx.SPOTWELD_HEXA1	Connection spotweld type is one hexa solid/spotweld layer.
Conx.SPOTWELD_HEXA12	Connection spotweld type is twelve hexa solids/spotweld layer.
Conx.SPOTWELD_HEXA16	Connection spotweld type is sixteen hexa solids/spotweld layer.
Conx.SPOTWELD_HEXA2	Connection spotweld type is two hexa solids/spotweld layer.
Conx.SPOTWELD_HEXA3	Connection spotweld type is three hexa solids/spotweld layer.
Conx.SPOTWELD_HEXA4	Connection spotweld type is four hexa solids/spotweld layer.
Conx.SPOTWELD_HEXA8	Connection spotweld type is eight hexa solids/spotweld layer.
Conx.SPOTWELD_MIG	Connection spotweld type is (beam) MIG weld.

Conx.SPOTWELD_SOLID1	This constant is deprecated in version 22.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Connection spotweld type is one hexa solid/spotweld layer. Use Conx.SPOTWELD_HEXA1 instead. [deprecated]
Conx.SPOTWELD_SOLID12	This constant is deprecated in version 22.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Connection spotweld type is twelve hexa solids/spotweld layer. Use Conx.SPOTWELD_HEXA12 instead. [deprecated]
Conx.SPOTWELD_SOLID16	This constant is deprecated in version 22.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Connection spotweld type is sixteen hexa solids/spotweld layer. Use Conx.SPOTWELD_HEXA16 instead. [deprecated]
Conx.SPOTWELD_SOLID4	This constant is deprecated in version 22.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Connection spotweld type is four hexa solids/spotweld layer. Use Conx.SPOTWELD_HEXA4 instead. [deprecated]
Conx.SPOTWELD_SOLID8	This constant is deprecated in version 22.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Connection spotweld type is eight hexa solids/spotweld layer. Use Conx.SPOTWELD_HEXA8 instead. [deprecated]

Conx properties

Name	Type	Description
adhesive_esize	real	Element size along the length of the adhesive run
adhesive_nelem	integer	The number of elements across the width of the adhesive
adhesive_width	real	The width of the adhesive run
angtol	real	angle tolerance for bolt
angtol2	real	angle tolerance at end 2 for 2 point bolt
assembly	integer/string	Assembly used to specify panels connection together, rather than individual layers. Integer for a part set ID, string for a PRIMER assembly (name).
assembly_type	constant	The assembly type. Can be Conx.PART_SET or Conx.ASSEMBLY.
colour	Colour	The colour of the connection
diameter	real	Diameter of spotweld/rigid
diameter2	real	Diameter of rigid at end 2
edge_distance	real	Spotweld line edge distance
edge_lock	logical	true if a spotweld line is locked to an edge, false if not
error (read only)	string	Description of the error if the connection cannot be made
error_details (read only)	string	Details of the error if the connection cannot be made

exists (read only)	logical	true if connection exists, false if referred to but not defined.
fit	integer	contact fitting method for library bolts
id	integer	Conx number. Also see the label property which is an alternative name for this.
include	integer	The Include file number that the connection is in.
label	integer	Conx number. Also see the id property which is an alternative name for this.
layers	integer	The number of layers the connection has.
length	real	Length of 1 point bolt, max thickness for 2 point bolt
length2	real	max thickness at end 2 for 2 point bolt
material	integer	The ID of the Material used for 'merge' bolt connections. i.e. Conx.BOLT_MRG_CYL, Conx.BOLT_MRG_CYL_BEAM,
model (read only)	integer	The Model number that the connection is in.
module	string	name of library module for bolt
part	integer	The ID of the Part used for adhesive or spotweld connections. Note that in v11.0 and above you are able to specify a different part IDs for elements in the connection between different layers. If you only have one part for the elements in the connection, then this is the value of this property. If there is more than one used, then the value of this property is the first part. If you set this property to a new value, then the all the elements in the connection will have this new part ID when it is realized. To set and retrieve information on parts used between different layers, the functions GetPidData() and SetPidData() should be used.
patch_coords	integer	The number of patch coordinate points the connection has (Adhesive patch only).
patch_topol	integer	The number of patch topology entries the connection has (Adhesive patch only).
path	integer	The number of path points the connection has (Adhesive only). Note that these points do **NOT** include the start and end points for the adhesive run. These are defined using the properties x, y, z and x2, y2, z2
pitch	real	Spotweld line pitch
resize	integer	snap to points fitting method for library bolts
saved_ settings	boolean	Whether settings are saved for a connection or not
shape	integer	shape for bolt attachment
shape2	integer	shape for bolt attachment at end 2 for 2 point bolt
spr2_id (read only)	integer	Internal label of C_SPR2 which applied to this rivet connection
spr2_match	boolean	True to use matching C_SPR2 for this rivet. False to create new C_SPR2 for each rivet. IF unset, a new C_SPR2 will be created.
spr2_ unshared (read only)	boolean	True if C_SPR2 is unique for this rivet
status	constant	The status of the connection. Can be Conx.DORMANT, Conx.MADE, Conx.INVALID, Conx.REALIZED or Conx.BAD.

subtype	constant	The connection subtype. For SPOTWELD and SPOTWELD_LINE connections the subtype can be: • Conx.SPOTWELD_BEAM • Conx.SPOTWELD_MIG • Conx.SPOTWELD_HEXA1 • Conx.SPOTWELD_HEXA2 • Conx.SPOTWELD_HEXA3 • Conx.SPOTWELD_HEXA4 • Conx.SPOTWELD_HEXA8 • Conx.SPOTWELD_HEXA12 • Conx.SPOTWELD_HEXA16 For BOLT connections the subtype can be: • Conx.BOLT_MRG_CYL • Conx.BOLT_MRG_CYL_BEAM • Conx.BOLT_MRG_CYL_BALL • Conx.BOLT_MRG_2PTS • Conx.BOLT_MRG_2PTS_RB • Conx.BOLT_MRG_2PTS_RJ • Conx.BOLT_MRG_CYL • Conx.BOLT_NRB_CYL_BEAM • Conx.BOLT_NRB_CYL_BALL • Conx.BOLT_NRB_SPH • Conx.BOLT_NRB_SPH_BALL • Conx.BOLT_NRB_SPH_DISC • Conx.BOLT_NRB_2PTS • Conx.BOLT_MODULE For ADHESIVE connections the subtype can be: Conx.ADHESIVE_SOLID. Conx.ADHESIVE_PATCH.
title	string	Title for connection
transparency	integer	The transparency of the connection (0-100) 0% is opaque, 100% is transparent.
type	constant	The connection type. Can be Conx.SPOTWELD, Conx.BOLT or Conx.RIVET or Conx.ADHESIVE.
user_data	string	User data for connection
x	real	X coordinate
x2	real	X coordinate for second point (adhesive only)
y	real	Y coordinate
y2	real	Y coordinate for second point (adhesive only)
z	real	Z coordinate
z2	real	Z coordinate for second point (adhesive only)

Detailed Description

The Conx class allows you to create, modify, edit and manipulate connections. See the documentation below for more details.

Constructor

new Conx(Model*[Model]*, x*[real]*, y*[real]*, z*[real]*, type (optional)*[constant]*, subtype (optional)*[constant]*, title (optional)*[string]*)

Description

Create a new Conx object.

Arguments

- **Model** (Model)

Model that connection will be created in

- **x** (real)

X coordinate

- **y** (real)

Y coordinate

- **z** (real)

Z coordinate

- **type (optional)** (constant)

Type of connection. Can be Conx.SPOTWELD, Conx.BOLT, Conx.ADHESIVE Conx.SPOTWELD_LINE or Conx.RIVET. If omitted type will be set to Conx.SPOTWELD.

- **subtype (optional)** (constant)

Subtype of connection. See property subtype for valid values. If omitted subtype will be set to the default subtype for this type of connection.

- **title (optional)** (string)

Title for the connection

Returns

Conx object

Return type

Conx

Example

To create a new connection in model m, at coordinates (20, 40, 10)

```
var c = new Conx(m, 20, 40, 10);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a connection.

Arguments

- **Comment** (Comment)

Comment that will be attached to the connection

Returns

No return value

Example

To associate comment c to the connection c:

```
c.AssociateComment(c);
```

Blank()

Description

Blanks the connection

Arguments

No arguments

Returns

No return value

Example

To blank connection c:

```
c.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the connections in the model.

Arguments

- **Model** (Model)

Model that all connections will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the connections in model m:

```
Conx.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged connections in the model.

Arguments

- **Model** (Model)

Model that all the flagged connections will be blanked in

- **flag** (Flag)

Flag set on the connections that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the connections in model m flagged with f:

```
Conx.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the connection is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if connection c is blanked:

```
if (c.Blanked() ) do_something...
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the connection.

Arguments

- **flag** (Flag)

Flag to clear on the connection

Returns

No return value

Example

To clear flag f for connection c:

```
c.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the connection. The target include of the copied connection can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

Conx object

Return type

Conx

Example

To copy connection c into connection z:

```
var z = c.Copy();
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a connection.

Arguments

- **Comment** (Comment)

Comment that will be detached from the connection

Returns

No return value

Example

To detach comment c from the connection c:

```
c.DetachComment(c);
```

EmptyPatch()

Description

Empties the patch topology/coordinates data.

Arguments

No arguments

Returns

No return value.

Example

To empty the patch topology/coordinates data for connection c;

```
c.EmptyPatch();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for connection with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The connection object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for connection c:

```
c.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for connection c:

```
c.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for connection. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for connection c:

```
c.Error("My custom error");
```

ExtractColour()

Description

Extracts the **actual** colour used for connection.
By default in PRIMER many entities such as elements get their colour automatically from the part that they are in. PRIMER cycles through 13 default colours based on the label of the entity. In this case the connection colour property will return the value Colour.PART instead of the actual colour. This method will return the actual colour which is used for drawing the connection.

Arguments

No arguments

Returns

colour value (integer)

Return type

Number

Example

To return the colour used for drawing connection c:

```
var colour = c.ExtractColour();
```

First(Model*[Model]*) [static]

Description

Returns the first connection in the model.

Arguments

- **Model** (Model)

Model to get first connection in

Returns

Conx object (or null if there are no connections in the model).

Return type

Conx

Example

To get the first connection in model m:

```
var c = Conx.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free connection label in the model. Also see Conx.LastFreeLabel(), Conx.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free connection label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

Conx label.

Return type

Number

Example

To get the first free connection label in model m:

```
var label = Conx.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the connections in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all connections will be flagged in

- **flag** (Flag)

Flag to set on the connections

Returns

No return value

Example

To flag all of the connections with flag f in model m:

```
Conx.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the connection is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the connection

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if connection c has flag f set on it:

```
if (c.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each connection in the model.
Note that ForEach has been designed to make looping over connections as fast as possible and so has some limitations.
Firstly, a single temporary Conx object is created and on each function call it is updated with the current connection data. This means that you should not try to store the Conx object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new connections inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all connections are in

- **func** (function)

Function to call for each connection

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the connections in model m:

```
Conx.ForEach(m, test);
function test(c)
{
// c is Conx object
}
```

To call function test for all of the connections in model m with optional object:

```
var data = { x:0, y:0 };
Conx.ForEach(m, test, data);
function test(c, extra)
{
// c is Conx object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Conx objects or properties for all of the connections in a model in PRIMER. If the optional property argument is not given then an array of Conx objects is returned. If the property argument is given, that property value for each connection is returned in the array instead of a Conx object

Arguments

- **Model** (Model)

Model to get connections from

- **property (optional)** (string)

Name for property to get for all connections in the model

Returns

Array of Conx objects or properties

Return type

Array

Example

To make an array of Conx objects for all of the connections in model m:

```
var a = Conx.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each connection in model m:

```
var a = Conx.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a connection.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the connection c:

```
var comm_array = c.GetComments();
```

GetElements()

Description

Returns the beams/solids that are used in the connection weld.

Arguments

No arguments

Returns

An array containing the element IDs (or null if no elements).

Return type

Array

Example

To get the elements for connection c:

```
var elems = c.GetElements();
```

GetEntities(type*[string]*)

Description

Returns list of the entities of type that are used in the connection.

Arguments

- **type** (string)

The type of the item in the reference list (for a list of types see Appendix I of the PRIMER manual).

Returns

An array containing the item IDs (or null if none).

Return type

Array

Example

To get list of nodes for connection c:

```
var items = c.GetEntities("NODE");
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of Conx objects for all of the flagged connections in a model in PRIMER If the optional property argument is not given then an array of Conx objects is returned. If the property argument is given, then that property value for each connection is returned in the array instead of a Conx object

Arguments

- **Model** (Model)

Model to get connections from

- **flag** (Flag)

Flag set on the connections that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged connections in the model

Returns

Array of Conx objects or properties

Return type

Array

Example

To make an array of Conx objects for all of the connections in model m flagged with f:

```
var c = Conx.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the connections in model m flagged with f:

```
var a = Conx.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the Conx object for a connection ID.

Arguments

- **Model** (Model)

Model to find the connection in

- **number** (integer)

number of the connection you want the Conx object for

Returns

Conx object (or null if connection does not exist).

Return type

Conx

Example

To get the Conx object for connection 100 in model m

```
var c = Conx.GetFromID(m, 100);
```

GetLayerData(layer*[integer]*)

Description

Returns the data for a layer of the connection.

Arguments

- **layer** (integer)

The layer you want the data for. **Note that layers start at 0, not 1**.

Returns

An array containing the layer data.

Return type

Array

Example

To get the data for the 3rd layer for connection c:

```
var l_data = c.GetLayerData(2);
```

GetLayerShells(layer*[integer]*)

Description

Returns the attached shells for a layer of the connection.

Arguments

- **layer** (integer)

The layer you want the data for. **Note that layers start at 0, not 1**.

Returns

Array of Shell objects or null if not valid

Return type

Array

Example

To get the attached shells for the 3rd layer for connection c:

```
var shells = c.GetLayerShells(2);
```

GetParameter(prop*[string]*)

Description

Checks if a Conx property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the Conx.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

connection property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if Conx property c.example is a parameter:

```
Options.property_parameter_names = true;
if (c.GetParameter(c.example) ) do_something...
Options.property_parameter_names = false;
```

To check if Conx property c.example is a parameter by using the GetParameter method:

```
if (c.ViewParameters().GetParameter(c.example) ) do_something...
```

GetPatchCoords(point*[integer]*)

Description

Returns the data for a patch coordinate of an adhesive patch connection.

Arguments

- **point** (integer)

The point you want the data for. **Note that points start at 0, not 1**.

Returns

An array containing the patch coordinate.

Return type

Array

Example

To get the data for the 3rd patch coordinate for connection c:

```
var p_data = c.GetPatchCoords(2);
```

GetPatchTopol(point*[integer]*)

Description

Returns the topology for a patch quad/tria of an adhesive patch connection.

Arguments

- **point** (integer)

The patch quad/tria you want the data for. **Note that points start at 0, not 1**.

Returns

Array of numbers containing the patch topology information.

Return type

Number

Example

To get the data for the 3rd patch quad/tria for connection c:

```
var p_data = c.GetPatchTopol(2);
```

GetPathData(point*[integer]*)

Description

Returns the data for a path point of an adhesive/spotweld line connection.

Arguments

- **point** (integer)

The point you want the data for. **Note that points start at 0, not 1**.

Returns

An array containing the path data.

Return type

Array

Example

To get the data for the 3rd path point for connection c:

```
var p_data = c.GetPathData(2);
```

GetPidData()

Description

Returns an array of Part objects for the connection FE entities. A connection can contain elements with different part ID’s between different layers. If one part ID is returned, that part is used for all elements in the connection. Not applicable for bolts.

Arguments

No arguments

Returns

Array of Part objects

Return type

Array

Example

To make an array of Part objects for connection c

```
var arr = c.GetPidData();
```

GetSettings()

Description

Returns an object of settings stored with the connection.

Arguments

No arguments

Returns

Object with the following properties:

Name	Type	Description
angle_tolerance	real	Angle tolerance
bolt_adjust_mass	boolean	Adjust bolt mass when creating bolt entitites
bolt_dth_beam	boolean	Add database history beam when bolt beam is created
bolt_feature_line	boolean	Consider feature line for bolt holes
bolt_nrb_min_mass	real	Bolt rigid NRB minimum mass
bolt_part_min_mass	real	Bolt rigid part minimum mass
clinch	boolean	Allow connections to join a clinch type connection
consistent_weld_area	boolean	Turn on consistent weld area so multihex welds are pi*d*d/4
edge_distance	real	Edge distance
glue_break_angle	real	Glue break angle
glue_hard_aspect	real	Glue hard aspect ratio
glue_soft_aspect	real	Glue soft aspect ratio
length_check	boolean	Check length

max_length	real	Maximum length
max_panels	integer	Maximum number of panels
max_warpage	real	Maximum warpage
migweld_pitch	boolean	Create migwelds at fixed pitch (without nodal attachment)
min_length	real	Minimum length
nrb_arcweld	boolean	Create arcweld using NRBs (rather than contact)
panel_check	boolean	Check for maximum number of panels
patch_angle	real	Patch angle setting
patch_angle_check	boolean	Turn on or off patch angle check
same_part	boolean	Allow connections to join a part to itself
solid_free_edges	boolean	Consider free edges when orienting single solid spotwelds
spot_line_tol	real	Spotweld line search tolerance
spot_thickness	real	Search thickness
total_length	real	Total length
use_pid	boolean	Use _PID for beam connections
warpage_check	boolean	Check warpage value

Return type

object

Example

To make an Oject containing the stored settings of connection c

```
var o = Conx.GetSettings();
```

GetShellThickness(Layer*[integer]*)

Description

Returns an array containing a number of objects equal to the number of solid elements in the connection. Each object contains the corresponding solid element object, and shell element objects and their thicknesses. The argument allows the user to output only shells from all layers, or a particular layer. **Note that a carriage return is not added**.

Arguments

- **Layer** (integer)

ID of the connection layer containing the shells from which the thicknesses will be extracted. If a value of zero or lower is input, all layers will be considered in the output data.

Returns

An array containing a number of objects equal to the number of solid elements in the connection. Each object has the following properties:

Name	Type	Description
shell1	Shell	Shell object in the specified layer in contact with the 1st node of the solid adhesive element
shell2	Shell	Shell object in the specified layer in contact with the 2nd node of the solid adhesive element

shell3	Shell	Shell object in the specified layer in contact with the 3rd node of the solid adhesive element
shell4	Shell	Shell object in the specified layer in contact with the 4th node of the solid adhesive element
shell5	Shell	Shell object in the specified layer in contact with the 5th node of the solid adhesive element
shell6	Shell	Shell object in the specified layer in contact with the 6th node of the solid adhesive element
shell7	Shell	Shell object in the specified layer in contact with the 7th node of the solid adhesive element
shell8	Shell	Shell object in the specified layer in contact with the 8th node of the solid adhesive element
solid	Solid	Solid object associated with the shells in the return object.
sthk1	real	Thickness of shell in the specified layer in contact with the 1st node of the solid adhesive element
sthk2	real	Thickness of shell in the specified layer in contact with the 2nd node of the solid adhesive element
sthk3	real	Thickness of shell in the specified layer in contact with the 3rd node of the solid adhesive element
sthk4	real	Thickness of shell in the specified layer in contact with the 4th node of the solid adhesive element
sthk5	real	Thickness of shell in the specified layer in contact with the 5th node of the solid adhesive element
sthk6	real	Thickness of shell in the specified layer in contact with the 6th node of the solid adhesive element
sthk7	real	Thickness of shell in the specified layer in contact with the 7th node of the solid adhesive element
sthk8	real	Thickness of shell in the specified layer in contact with the 8th node of the solid adhesive element

Return type

object

Example

To get the data for connection c layer 1:

```
var sThkArr = c.GetShellThickness(1); // sThkArr is an array of objects
containing shells and shell thicknesses in only layer 1 for each solid element
in the connection.
var st0_1 = sThkArr[0].sthk1;         // st0_1 is the shell thickness of the
shell attached to node 1 in the first solid element (array index 0) in the
sThkArr array.
var sh1_4 = sThkArr[1].shell4;        // sth1_4 is the shell object
corresponding to node 1 in the second solid element (array index 1) in the
sThkArr array.
var so4   = sThkArr[4].solid;         // so4 is the fifth solid element object
(array index 4) in the sThkArr array.
```

Keyword()

Description

Returns the keyword for this connection (*CONNECTION_START_SPOTWELD etc). **Note that a carriage return is not added**. See also Conx.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for connection c:

```
var key = c.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the connection. **Note that a carriage return is not added**. See also Conx.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for connection c:

```
var cards = c.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last connection in the model.

Arguments

- **Model** (Model)

Model to get last connection in

Returns

Conx object (or null if there are no connections in the model).

Return type

Conx

Example

To get the last connection in model m:

```
var c = Conx.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free connection label in the model. Also see Conx.FirstFreeLabel(), Conx.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free connection label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

Conx label.

Return type

Number

Example

To get the last free connection label in model m:

```
var label = Conx.LastFreeLabel(m);
```

Next()

Description

Returns the next connection in the model.

Arguments

No arguments

Returns

Conx object (or null if there are no more connections in the model).

Return type

Conx

Example

To get the connection in model m after connection c:

```
var c = c.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) connection label in the model. Also see Conx.FirstFreeLabel(), Conx.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free connection label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

Conx label.

Return type

Number

Example

To get the next free connection label in model m:

```
var label = Conx.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a connection.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only connections from that model can be picked. If the argument is a Flag then only connections that are flagged with *limit* can be selected. If omitted, or null, any connections from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

Conx object (or null if not picked)

Return type

Conx

Example

To pick a connection from model m giving the prompt 'Pick connection from screen':

```
var c = Conx.Pick('Pick connection from screen', m);
```

Previous()

Description

Returns the previous connection in the model.

Arguments

No arguments

Returns

Conx object (or null if there are no more connections in the model).

Return type

Conx

Example

To get the connection in model m before connection c:

```
var c = c.Previous();
```

RealizeAll(Model*[Model]*) [static]

Description

Realizes all of the connections in the model.

Arguments

- **Model** (Model)

Model that all connections will be realized in

Returns

No return value

Example

To realize all of the connections in model m:

```
Conx.RealizeAll(m);
```

RealizeFlagged(Model*[Model]*, flag*[Flag]*) [static]

Description

Realizes all of the flagged connections in the model.

Arguments

- **Model** (Model)

Model that the flagged connections will be realized in

- **flag** (Flag)

Flag set on the connections that you want to realize

Returns

No return value

Example

To realize all of the connections in model m flagged with f:

```
Conx.RealizeFlagged(m, f);
```

ReloadConnectors() [static]

Description

Reload all modules from primer_library/connectors

Arguments

No arguments

Returns

No return value

Example

```
Conx.ReloadConnectors();
```

RemovePatchTopol(layer*[integer]*)

Description

Deletes the topology at a particular location for patch type adhesive.

Arguments

- **layer** (integer)

The topology location you want to remove. **Note that layers start at 0, not 1**.

Returns

No return value.

Example

To remove the 3rd topology data for connection c:

```
c.RemovePatchTopol(2);
```

RemovePathData(layer*[integer]*)

Description

Deletes a pathc point for a line adhesive connection.

Arguments

- **layer** (integer)

The point you want to remove. **Note that layers start at 0, not 1**.

Returns

No return value.

Example

To remove the 3rd point from connection c:

```
c.RemovePathData(2);
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the connections in the model.

Arguments

- **Model** (Model)

Model that all connections will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the connections in model m, from 1000000:

```
Conx.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged connections in the model.

Arguments

- **Model** (Model)

Model that all the flagged connections will be renumbered in

- **flag** (Flag)

Flag set on the connections that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the connections in model m flagged with f, from 1000000:

```
Conx.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select connections using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting connections

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only connections from that model can be selected. If the argument is a Flag then only connections that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any connections can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted

the selection will be modal.

Returns

Number of connections selected or null if menu cancelled

Return type

Number

Example

To select connections from model m, flagging those selected with flag f, giving the prompt 'Select connections':

```
Conx.Select(f, 'Select connections', m);
```

To select connections, flagging those selected with flag f but limiting selection to connections flagged with flag l, giving the prompt 'Select connections':

```
Conx.Select(f, 'Select connections', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the connection.

Arguments

- **flag** (Flag)

Flag to set on the connection

Returns

No return value

Example

To set flag f for connection c:

```
c.SetFlag(f);
```

SetLayerData(layer*[integer]*, items*[integer or string]*)

Description

Sets the data for a layer of the connection.

Arguments

- **layer** (integer)

The layer you want to set the data for. **Note that layers start at 0, not 1**.

- **items** (integer or string)

The items for the layer definition. As layer definitions can be part IDs, part names, CAD names, part set IDs, part set names or assemby names the following logic is used. If the item is an integer it is assumed to be a part ID. If the item is a string then it must be in the format 'P<part ID>', 'P:<part name>', 'C:<CAD name>', 'S<set ID>', 'S:<set name>' or 'A:<assembly name>'. Each item must be of the same type as the first item, e.g. if the first item is a part ID, then all items must be a part ID (it cannot be a part name etc).

This argument can be repeated if required
Alternatively a single array argument containing the multiple values can be given

Returns

No return value.

Example

To set the data for the 3rd layer for connection c, to be part IDs 10 and 20:

```
c.SetLayerData(2, 10, 20);
```

or

```
var a = new Array(10, 20);
c.SetLayerData(2, a);
```

SetPatchCoords(point*[integer]*, x*[real]*, y*[real]*, z*[real]*)

Description

Sets a coordinate used by the adhesive patch connection type.

Arguments

- **point** (integer)

The point you want to set the data for. **Note that points start at 0, not 1**.

- **x** (real)

X coordinate of point

- **y** (real)

Y coordinate of point

- **z** (real)

Z coordinate of point

Returns

No return value.

Example

To set the position for the 3rd patch point for connection c, to be (10, 20, 30);

```
c.SetPatchCoords(2, 10, 20, 30);
```

SetPatchTopol(point*[integer]*, c1*[integer]*, c2*[integer]*, c3*[integer]*, c4 (optional)*[integer]*)

Description

Sets the topology used by the adhesive patch connection type.

Arguments

- **point** (integer)

The point you want to set the data for. **Note that points start at 0, not 1**.

- **c1** (integer)

1st coordinate location point

- **c2** (integer)

2nd coordinate location point

- **c3** (integer)

3rd coordinate location point

- **c4 (optional)** (integer)

4th coordinate location point

Returns

No return value.

Example

To set the topology for the 3rd patch quad/tria for connection c, to be (1, 4, 3, 6);

```
c.SetPatchTopol(2, 1, 4, 3, 6);
```

SetPathData(point*[integer]*, x*[real]*, y*[real]*, z*[real]*)

Description

Sets the data for a path point of the connection.

Arguments

- **point** (integer)

The point you want to set the data for. **Note that points start at 0, not 1**.

- **x** (real)

X coordinate of point

- **y** (real)

Y coordinate of point

- **z** (real)

Z coordinate of point

Returns

No return value.

Example

To set the position for the 3rd path point for connection c, to be (10, 20, 30);

```
c.SetPathData(2, 10, 20, 30);
```

SetPidData(items*[integer or string]*)

Description

Sets the element part IDs for the connection. A different part can be defined for elements in the connection between different layers. Not applicable for bolts.

Arguments

- **items** (integer or string)

Part labels of the items in the PID layer list.

This argument can be repeated if required
Alternatively a single array argument containing the multiple values can be given

Returns

No return value.

Example

To set the part data for c, to be part IDs 10 and 20:

```
c.SetPidData(10, 20);
```

or

```
var a = new Array(10, 20);
c.SetPidData(a);
```

SetRuleDiameter(diameter*[integer]*) [static]

Description

Set the diameter for a spotweld ring when running a rule. Note that this method can only be called when running a connection rule script. It will not have any effect if used in a 'normal' script.

Arguments

- **diameter** (integer)

The diameter to set for the ring

Returns

No return value

Example

To set the diameter for a ring to be 10.0:

```
Conx.SetRuleDiameter(10.0);
```

SetRuleFEPID(pid*[integer]*) [static]

Description

Set the PID for spotweld beam/solid elements or adhesive solids when running a rule. Note that this method can only be called when running a connection rule script. It will not have any effect if used in a 'normal' script.

Arguments

- **pid** (integer)

The PID to set for the spotweld or adhesive elements

Returns

No return value

Example

To set the PID for a spotweld to be 1000:

```
Conx.SetRuleFEPID(1000);
```

SetRulePID(pid*[integer]*) [static]

Description

Set the PID for a spotweld ring when running a rule. Note that this method can only be called when running a connection rule script. It will not have any effect if used in a 'normal' script.

Arguments

- **pid** (integer)

The PID to set for the ring

Returns

No return value

Example

To set the PID for a ring to be 1000:

```
Conx.SetRulePID(1000);
```

SetSettings(data*[object]*)

Description

Sets the settings stored on a connection entity. Not applicable for bolts.

Arguments

- **data** (object)

Object containing the connection settings data. The properties can be:

Object has the following properties:

Name	Type	Description
angle_tolerance (optional)	real	Angle tolerance
bolt_adjust_mass (optional)	boolean	Adjust bolt mass when creating bolt entities
bolt_dth_beam (optional)	boolean	Add database history beam when bolt is created
bolt_feature_line (optional)	boolean	Consider feature line for bolt holes
bolt_nrb_min_mass (optional)	real	Bolt NRB minimum mass
bolt_part_min_mass (optional)	real	Bolt rigid part minimum mass
clinch (optional)	boolean	Allow connections to join a clinch type connection
consistent_weld_area (optional)	boolean	Use consistent area for multihex welds
edge_distance (optional)	real	Edge distance
glue_break_angle (optional)	real	Glue break angle
glue_hard_aspect (optional)	real	Glue hard aspect ratio
glue_soft_aspect (optional)	real	Glue soft aspect ratio
length_check (optional)	boolean	Check the connection length
max_length (optional)	real	Maximum length of connection
max_panels (optional)	integer	Maximum number of panels
max_warpage (optional)	real	Maximum warpage
migweld_pitch (optional)	boolean	Create migwelds at fixed pitch (without nodal attachment)
min_length (optional)	real	Minimum length of connection
nrb_arcweld (optional)	boolean	Create arcweld using NRBs (rather than contact)
panel_check (optional)	boolean	Check for maximum number of panels
patch_angle (optional)	real	Patch angle

patch_angle_check (optional)	boolean	Check the patch angle
same_part (optional)	boolean	Allow connections to join a part to itself
solid_free_edges (optional)	boolean	Consider free edges when orienting single solid spotwelds
spot_line_tol (optional)	real	Spotweld line search tolerance
spot_thickness (optional)	real	Search thickness
total_length (optional)	real	Total length of connection
use_pid (optional)	boolean	Use _PID for beam connections
warpage_check (optional)	boolean	Check warpage value

Returns

No return value.

Example

To set the various settings for a connection c:

```
var data = { length_check:true, total_length:1.5, warpage_check:false, angle_
tolerance:5.0 };
c.SetSettings(data);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the connection. The connection will be sketched until you either call Conx.Unsketch(), Conx.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the connection is sketched. If omitted redraw is true. If you want to sketch several connections and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch connection c:

```
c.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged connections in the model. The connections will be sketched until you either call Conx.Unsketch(), Conx.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged connections will be sketched in

- **flag** (Flag)

Flag set on the connections that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the connections are sketched. If omitted redraw is true. If you want to sketch flagged connections several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all connections flagged with flag in model m:

```
Conx.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of connections in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing connections should be counted. If false or omitted referenced but undefined connections will also be included in the total.

Returns

number of connections

Return type

Number

Example

To get the total number of connections in model m:

```
var total = Conx.Total(m);
```

Unblank()

Description

Unblanks the connection

Arguments

No arguments

Returns

No return value

Example

To unblank connection c:

```
c.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the connections in the model.

Arguments

- **Model** (Model)

Model that all connections will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the connections in model m:

```
Conx.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged connections in the model.

Arguments

- **Model** (Model)

Model that the flagged connections will be unblanked in

- **flag** (Flag)

Flag set on the connections that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the connections in model m flagged with f:

```
Conx.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the connections in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all connections will be unset in

- **flag** (Flag)

Flag to unset on the connections

Returns

No return value

Example

To unset the flag f on all the connections in model m:

```
Conx.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the connection.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the connection is unsketched. If omitted redraw is true. If you want to unsketch several connections and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch connection c:

```
c.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all connections.

Arguments

- **Model** (Model)

Model that all connections will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the connections are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all connections in model m:

```
Conx.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged connections in the model.

Arguments

- **Model** (Model)

Model that all connections will be unsketched in

- **flag** (Flag)

Flag set on the connections that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the connections are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all connections flagged with flag in model m:

```
Conx.UnsketchAll(m, flag);
```

UseParentLayer(option*[boolean]*) [static]

Description

True (default) means put bolt FE into parent layer where possible.

Arguments

- **option** (boolean)

True (default) means put bolt FE into parent layer where possible.

Returns

No return value

Example

To switch off use of parent layer (and use current layer)

```
Conx.UseParentLayer(false);
```

UseSPR2Pref(option*[boolean]*) [static]

Description

True (default) means use the pref settings for C_SPR2 created when rivet realized.

Arguments

- **option** (boolean)

True (default) means use the pref settings for C_SPR2 created when rivet realized.

Returns

No return value

Example

To ignore any pref settings and use zero for newly created C_SPR2 cards

```
Conx.UseSPR2Pref(false);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

Conx object.

Return type

Conx

Example

To check if Conx property c.example is a parameter by using the Conx.GetParameter() method:

```
if (c.ViewParameters().GetParameter(c.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for connection with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The connection object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)

details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for connection c:

```
c.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for connection c:

```
c.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for connection. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for connection c:

```
c.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this connection.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for connection c:

```
var xrefs = c.Xrefs();
```

---

toString()

Description

Creates a string containing the connection data in keyword format. Note that this contains the keyword header and the keyword cards. See also Conx.Keyword() and Conx.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for connection n in keyword format

```
var s = c.toString();
```

Dummy class

The Dummy class gives you access to dummy cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetAssembly(index*[integer]*)
- GetAssemblyChildInfo(label*[integer]*, index*[integer]*)
- GetAssemblyFromID(label*[integer]*)
- GetAssemblyPart(label*[integer]*)
- GetComments()
- GetParameter(prop*[string]*)
- GetPoint(index*[integer]*)
- GetPointData(rpt*[integer]*)
- GetPointTitle(rpt*[integer]*)
- Next()
- Previous()
- RemovePoint(index*[integer]*)
- SelectAssembly()
- SetAssemblyNodeSet(label*[integer]*, nsid*[integer]*)
- SetAssemblyPart(label*[integer]*, pid*[integer]*)
- SetAssemblyPartSet(label*[integer]*, psid*[integer]*)

- SetAssemblyStopAngle(label*[integer]*, axis*[integer]*, stop_neg*[real]*, stop_pos*[real]*)
- SetFlag(flag*[Flag]*)
- SetPoint(index*[integer]*, data*[object]*)
- Sketch(redraw (optional)*[boolean]*)
- SketchAssembly(label*[integer]*, redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- UnsketchAssembly(label*[integer]*, redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()

Dummy properties

Name	Type	Description
assemblies (read only)	integer	Number of assemblies defined.
exists (read only)	logical	true if dummy exists, false if referred to but not defined.
id (read only)	integer	Dummy number. Also see the label property which is an alternative name for this.
include	integer	The Include file number that the dummy is in.
label (read only)	integer	Dummy number. Also see the id property which is an alternative name for this.
model (read only)	integer	The Model number that the dummy is in.
points (read only)	integer	Number of reference points defined.
title	string	Dummy title.
xhpoint (read only)	real	H-Point X coordinate.
yhpoint (read only)	real	H-Point Y coordinate.
zhpoint (read only)	real	H-Point Z coordinate.

Detailed Description

The Dummy class allows you to create, modify, edit and manipulate dummy cards. See the documentation below for more details.

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a dummy.

Arguments

- **Comment** (Comment)

Comment that will be attached to the dummy

Returns

No return value

Example

To associate comment c to the dummy d:

```
d.AssociateComment(c);
```

Blank()

Description

Blanks the dummy

Arguments

No arguments

Returns

No return value

Example

To blank dummy d:

```
d.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the dummies in the model.

Arguments

- **Model** (Model)

Model that all dummies will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the dummies in model m:

```
Dummy.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged dummies in the model.

Arguments

- **Model** (Model)

Model that all the flagged dummies will be blanked in

- **flag** (Flag)

Flag set on the dummies that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the dummies in model m flagged with f:

```
Dummy.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the dummy is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if dummy d is blanked:

```
if (d.Blanked() ) do_something...
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the dummy.

Arguments

- **flag** (Flag)

Flag to clear on the dummy

Returns

No return value

Example

To clear flag f for dummy d:

```
d.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the dummy. The target include of the copied dummy can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

Dummy object

Return type

Dummy

Example

To copy dummy d into dummy z:

```
var z = d.Copy();
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a dummy.

Arguments

- **Comment** (Comment)

Comment that will be detached from the dummy

Returns

No return value

Example

To detach comment c from the dummy d:

```
d.DetachComment(c);
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for dummy with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The dummy object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...

autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for dummy d:

```
d.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for dummy d:

```
d.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for dummy. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for dummy d:

```
d.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first dummy in the model.

Arguments

- **Model** (Model)

Model to get first dummy in

Returns

Dummy object (or null if there are no dummies in the model).

Return type

Dummy

Example

To get the first dummy in model m:

```
var d = Dummy.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free dummy label in the model. Also see Dummy.LastFreeLabel(), Dummy.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free dummy label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

Dummy label.

Return type

Number

Example

To get the first free dummy label in model m:

```
var label = Dummy.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the dummies in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all dummies will be flagged in

- **flag** (Flag)

Flag to set on the dummies

Returns

No return value

Example

To flag all of the dummies with flag f in model m:

```
Dummy.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the dummy is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the dummy

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if dummy d has flag f set on it:

```
if (d.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each dummy in the model.
Note that ForEach has been designed to make looping over dummies as fast as possible and so has some limitations.
Firstly, a single temporary Dummy object is created and on each function call it is updated with the current dummy data. This means that you should not try to store the Dummy object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new dummies inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all dummies are in

- **func** (function)

Function to call for each dummy

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the dummies in model m:

```
Dummy.ForEach(m, test);
function test(d)
{
// d is Dummy object
}
```

To call function test for all of the dummies in model m with optional object:

```
var data = { x:0, y:0 };
Dummy.ForEach(m, test, data);
function test(d, extra)
{
// d is Dummy object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Dummy objects or properties for all of the dummies in a model in PRIMER. If the optional property argument is not given then an array of Dummy objects is returned. If the property argument is given, that property value for each dummy is returned in the array instead of a Dummy object

Arguments

- **Model** (Model)

Model to get dummies from

- **property (optional)** (string)

Name for property to get for all dummies in the model

Returns

Array of Dummy objects or properties

Return type

Array

Example

To make an array of Dummy objects for all of the dummies in model m:

```
var a = Dummy.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each dummy in model m:

```
var a = Dummy.GetAll(m, 'foo');
```

GetAssembly(index*[integer]*)

Description

Returns the information for an assembly

Arguments

- **index** (integer)

The index of the assembly you want the coordinates for. **Note that reference points start at 0, not 1**. 0 <= index < assemblies

Returns

Object with the following properties:

Name	Type	Description
label	integer	Assembly label
parent	integer	Parent assembly label
title	string	Assembly title

Return type

object

Example

To get the information for the 3rd assembly for dummy d:

```
var info = d.GetAssembly(2);
```

GetAssemblyChildInfo(label*[integer]*, index*[integer]*)

Description

Get information about a child assembly from its parent assembly.

Arguments

- **label** (integer)

The label of the parent assembly.

- **index** (integer)

index of the child (start with 0 till n-1, where n is total number of child).

Returns

Object with the following properties:

Name	Type	Description
dof_code	integer	Degree of freedom codes
joint_stiff	integer	Constrained joint stiffness label
label	integer	Label of the child assembly
node_a	integer	Node A label
node_b	integer	Node B label

Return type

object

Example

To get the information of first child assembly for which index will be 0 (starts with 0) in the assembly with label = 2 for dummy d:

```
var info = d.GetAssemblyChildInfo(2, 0);
```

GetAssemblyFromID(label*[integer]*)

Description

Get assembly information for a given assembly ID and returns an object containing the details.

Arguments

- **label** (integer)

The label of the assembly you want the Assembly object for.

Returns

Object with the following properties:

Name	Type	Description
child	integer	Total number of child assembly(s).
exists	logical	true if assembly exists, false if not defined.
index	integer	Assembly index
label	integer	Assembly label
parent	integer	Parent assembly label
rx	real	x-angle of the assembly.
ry	real	y-angle of the assembly.
rz	real	z-angle of the assembly.
title	string	Assembly title

Return type

object

Example

To get the information for the assembly with label = 2 for dummy d:

```
var info = d.GetAssemblyFromID(2);
```

GetAssemblyPart(label*[integer]*)

Description

Returns an array of Part objects representing all the parts within the assembly.

Arguments

- **label** (integer)

The label of the assembly.

Returns

Array of Part objects

Return type

Array

Example

To get all the parts in the assembly with label = 2 for dummy d:

```
var info = d.GetAssemblyPart(2);
```

GetComments()

Description

Extracts the comments associated to a dummy.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the dummy d:

```
var comm_array = d.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of Dummy objects for all of the flagged dummies in a model in PRIMER If the optional property argument is not given then an array of Dummy objects is returned. If the property argument is given, then that property value for each dummy is returned in the array instead of a Dummy object

Arguments

- **Model** (Model)

Model to get dummies from

- **flag** (Flag)

Flag set on the dummies that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged dummies in the model

Returns

Array of Dummy objects or properties

Return type

Array

Example

To make an array of Dummy objects for all of the dummies in model m flagged with f:

```
var d = Dummy.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the dummies in model m flagged with f:

```
var a = Dummy.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the Dummy object for a dummy ID.

Arguments

- **Model** (Model)

Model to find the dummy in

- **number** (integer)

number of the dummy you want the Dummy object for

Returns

Dummy object (or null if dummy does not exist).

Return type

Dummy

Example

To get the Dummy object for dummy 100 in model m

```
var d = Dummy.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a Dummy property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the Dummy.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

dummy property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if Dummy property d.example is a parameter:

```
Options.property_parameter_names = true;
if (d.GetParameter(d.example) ) do_something...
Options.property_parameter_names = false;
```

To check if Dummy property d.example is a parameter by using the GetParameter method:

```
if (d.ViewParameters().GetParameter(d.example) ) do_something...
```

GetPoint(index*[integer]*)

Description

Returns the information for a reference point

Arguments

- **index** (integer)

The index of the reference point you want the information for. **Note that reference points start at 0, not 1**. 0 <= index < points

Returns

Object with the following properties:

Name	Type	Description
assembly	integer	Assembly label
csys	integer	Coordinate system
hpt	boolean	If point has been automatically created by PRIMER at the H-point
label	integer	Point label
node	integer	Node label (0 if coordinate)
rx	boolean	Point restrained rotationally in X
ry	boolean	Point restrained rotationally in Y
rz	boolean	Point restrained rotationally in Z
title	string	Point title
tx	boolean	Point restrained translationally in X
ty	boolean	Point restrained translationally in Y
tz	boolean	Point restrained translationally in Z
x	real	Node/point x coordinate
y	real	Node/point y coordinate
z	real	Node/point z coordinate

Return type

object

Example

To get the information for the 3rd reference point for dummy d:

```
var info = d.GetPoint(2);
```

GetPointData(rpt*[integer]*)

Description

Returns the coordinates of a reference point

Arguments

- **rpt** (integer)

The reference point you want the coordinates for. **Note that reference points start at 0, not 1**.

Returns

Array containing the reference point coordinates

Return type

Array

Example

To get the coordinates of the 3rd reference point for dummy d:

```
var c = d.GetPointData(2);
```

GetPointTitle(rpt*[integer]*)

Description

Returns the title of a reference point

Arguments

- **rpt** (integer)

The reference point you want the title for. **Note that reference points start at 0, not 1**.

Returns

The reference point title

Return type

String

Example

To get the title of the 3rd reference point for dummy d:

```
var c = d.GetPointTitle(2);
```

Last(Model*[Model]*) [static]

Description

Returns the last dummy in the model.

Arguments

- **Model** (Model)

Model to get last dummy in

Returns

Dummy object (or null if there are no dummies in the model).

Return type

Dummy

Example

To get the last dummy in model m:

```
var d = Dummy.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free dummy label in the model. Also see Dummy.FirstFreeLabel(), Dummy.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free dummy label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

Dummy label.

Return type

Number

Example

To get the last free dummy label in model m:

```
var label = Dummy.LastFreeLabel(m);
```

Next()

Description

Returns the next dummy in the model.

Arguments

No arguments

Returns

Dummy object (or null if there are no more dummies in the model).

Return type

Dummy

Example

To get the dummy in model m after dummy d:

```
var d = d.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) dummy label in the model. Also see Dummy.FirstFreeLabel(), Dummy.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free dummy label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

Dummy label.

Return type

Number

Example

To get the next free dummy label in model m:

```
var label = Dummy.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a dummy.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only dummies from that model can be picked. If the argument is a Flag then only dummies that are flagged with *limit* can be selected. If omitted, or null, any dummies from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

Dummy object (or null if not picked)

Return type

Dummy

Example

To pick a dummy from model m giving the prompt 'Pick dummy from screen':

```
var d = Dummy.Pick('Pick dummy from screen', m);
```

Previous()

Description

Returns the previous dummy in the model.

Arguments

No arguments

Returns

Dummy object (or null if there are no more dummies in the model).

Return type

Dummy

Example

To get the dummy in model m before dummy d:

```
var d = d.Previous();
```

RemovePoint(index*[integer]*)

Description

Removes a reference point from a dummy

Arguments

- **index** (integer)

The index of the reference point you want to remove. **Note that reference points start at 0, not 1**. 0 <= index < points

Returns

no return value

Example

To remove for the 3rd reference point for dummy d:

```
d.RemovePoint(2);
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the dummies in the model.

Arguments

- **Model** (Model)

Model that all dummies will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the dummies in model m, from 1000000:

```
Dummy.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged dummies in the model.

Arguments

- **Model** (Model)

Model that all the flagged dummies will be renumbered in

- **flag** (Flag)

Flag set on the dummies that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the dummies in model m flagged with f, from 1000000:

```
Dummy.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select dummies using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting dummies

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only dummies from that model can be selected. If the argument is a Flag then only dummies that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any dummies can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of dummies selected or null if menu cancelled

Return type

Number

Example

To select dummies from model m, flagging those selected with flag f, giving the prompt 'Select dummies':

```
Dummy.Select(f, 'Select dummies', m);
```

To select dummies, flagging those selected with flag f but limiting selection to dummies flagged with flag l, giving the prompt 'Select dummies':

```
Dummy.Select(f, 'Select dummies', l);
```

SelectAssembly()

Description

Returns an array of objects containing the assembly informaitons or null if menu cancelled.

Arguments

No arguments

Returns

Array of objects with the following properties:

Name	Type	Description
label	integer	Assembly label
parent	integer	Parent assembly label
title	string	Assembly title

Return type

object

Example

To select assemblies in dummy d, giving the prompt "Select assemblies":

```
d.SelectAssembly();
```

SetAssemblyNodeSet(label*[integer]*, nsid*[integer]*)

Description

Sets a set node for a Dummy/HBM assembly.

Arguments

- **label** (integer)

The label of the assembly.

- **nsid** (integer)

The label of the set node to be added into the assembly.

Returns

no return value

Example

To set the node set with label=100 in the assembly with label = 2 for dummy d:

```
d.SetAssemblyNodeSet(2, 100);
```

SetAssemblyPart(label*[integer]*, pid*[integer]*)

Description

Sets a part for a Dummy assembly.

Arguments

- **label** (integer)

The label of the assembly.

- **pid** (integer)

The label of the set part to be added into the assembly.

Returns

no return value

Example

To set the part with label=100 in the assembly with label = 2 for dummy d:

```
d.SetAssemblyPart(2, 100);
```

SetAssemblyPartSet(label*[integer]*, psid*[integer]*)

Description

Sets a set part for a Dummy/HBM assembly.

Arguments

- **label** (integer)

The label of the assembly.

- **psid** (integer)

The label of the set part to be added into the assembly.

Returns

no return value

Example

To set the set part with label=100 in the assembly with label = 2 for dummy d:

```
d.SetAssemblyPartSet(2, 100);
```

SetAssemblyStopAngle(label*[integer]*, axis*[integer]*, stop_neg*[real]*, stop_pos*[real]*)

Description

Sets -ve and +ve stop angles in the assembly.

Arguments

- **label** (integer)

The label of the assembly.

- **axis** (integer)

Axis (0 = X, 1 = Y, or 2 = Z) on which to define stop angles.

- **stop_neg** (real)

-ve stop angle.

- **stop_pos** (real)

+ve stop angle.

Returns

No return value

Example

To set -90 and 90 stop angles in X-axis in the assembly with label = 2 for dummy d:

```
d.SetAssemblyStopAngle(2, 0, -90, 90);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the dummy.

Arguments

- **flag** (Flag)

Flag to set on the dummy

Returns

No return value

Example

To set flag f for dummy d:

```
d.SetFlag(f);
```

SetPoint(index*[integer]*, data*[object]*)

Description

Sets the data for a reference point in a dummy

Arguments

- **index** (integer)

The index of the reference point you want to set. **Note that reference points start at 0, not 1**. To add a new point use index points

- **data** (object)

Object containing the reference point data. The properties can be:

Object has the following properties:

Name	Type	Description
assembly	integer	Assembly label
csys (optional)	integer	Coordinate system label
node (optional)	integer	Node label (not rewuired if using x, y and z)
rx (optional)	boolean	Point restrained rotationally in X
ry (optional)	boolean	Point restrained rotationally in Y
rz (optional)	boolean	Point restrained rotationally in Z
title (optional)	string	Title
tx (optional)	boolean	Point restrained translationally in X
ty (optional)	boolean	Point restrained translationally in Y
tz (optional)	boolean	Point restrained translationally in Z
x (optional)	real	X coordinate (not required if using node)
y (optional)	real	Y coordinate (not required if using node)
z (optional)	real	Z coordinate (not required if using node)

Returns

no return value

Example

To add a new reference point to dummy d assembly 5 at node 1000 with title "Example point" restrained in x:

```
var data = { assembly:5, node:1000, title:"Example point", tx:true };
d.SetPoint(d.points, data);
```

To add a new reference point to dummy d assembly 5 at (10, 20, 30) with title "Example point":

```
var data = { assembly:5, x:10, y:20, z:30, title:"Example point" };
d.SetPoint(d.points, data);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the dummy. The dummy will be sketched until you either call Dummy.Unsketch(), Dummy.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the dummy is sketched. If omitted redraw is true. If you want to sketch several dummies and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch dummy d:

```
d.Sketch();
```

SketchAssembly(label*[integer]*, redraw (optional)*[boolean]*)

Description

Sketches the assembly

Arguments

- **label** (integer)

The label of the assembly you want to sketch.

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To sketch the assembly with label 3 in dummy d:

```
var info = d.SketchAssembly(3);
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged dummies in the model. The dummies will be sketched until you either call Dummy.Unsketch(), Dummy.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged dummies will be sketched in

- **flag** (Flag)

Flag set on the dummies that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the dummies are sketched. If omitted redraw is true. If you want to sketch flagged dummies several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all dummies flagged with flag in model m:

```
Dummy.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of dummies in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing dummies should be counted. If false or omitted referenced but undefined dummies will also be included in the total.

Returns

number of dummies

Return type

Number

Example

To get the total number of dummies in model m:

```
var total = Dummy.Total(m);
```

Unblank()

Description

Unblanks the dummy

Arguments

No arguments

Returns

No return value

Example

To unblank dummy d:

```
d.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the dummies in the model.

Arguments

- **Model** (Model)

Model that all dummies will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the dummies in model m:

```
Dummy.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged dummies in the model.

Arguments

- **Model** (Model)

Model that the flagged dummies will be unblanked in

- **flag** (Flag)

Flag set on the dummies that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the dummies in model m flagged with f:

```
Dummy.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the dummies in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all dummies will be unset in

- **flag** (Flag)

Flag to unset on the dummies

Returns

No return value

Example

To unset the flag f on all the dummies in model m:

```
Dummy.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the dummy.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the dummy is unsketched. If omitted redraw is true. If you want to unsketch several dummies and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch dummy d:

```
d.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all dummies.

Arguments

- **Model** (Model)

Model that all dummies will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the dummies are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all dummies in model m:

```
Dummy.UnsketchAll(m);
```

UnsketchAssembly(label*[integer]*, redraw (optional)*[boolean]*)

Description

Unsketches the assembly

Arguments

- **label** (integer)

The label of the assembly you want to unsketch.

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unsketch the assembly with label 3 in dummy d:

```
var info = d.UnsketchAssembly(3);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged dummies in the model.

Arguments

- **Model** (Model)

Model that all dummies will be unsketched in

- **flag** (Flag)

Flag set on the dummies that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the dummies are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all dummies flagged with flag in model m:

```
Dummy.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

Dummy object.

Return type

Dummy

Example

To check if Dummy property d.example is a parameter by using the Dummy.GetParameter() method:

```
if (d.ViewParameters().GetParameter(d.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for dummy with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The dummy object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for dummy d:

```
d.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for dummy d:

```
d.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for dummy. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for dummy d:

```
d.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this dummy.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for dummy d:

```
var xrefs = d.Xrefs();
```

File class

The File class allows you to read and write text files. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Copy(source*[string]*, dest*[string]*)
- Delete(filename*[string]*)
- DriveMapFilename(filename*[string]*, format*[constant]*)
- Exists(filename*[string]*)
- FindFiles(directory*[string]*, type (optional)*[constant]*)
- Get(url*[string]*, filename*[string]*, options (optional)*[object]*)
- IsAbsolute(filename*[string]*)
- IsDirectory(filename*[string]*)
- IsFile(filename*[string]*)
- IsReadable(filename*[string]*)
- IsWritable(filename*[string]*)
- Mkdir(directory*[string]*)
- Mktemp()
- Proxy(name*[string]*)
- ProxyPassword(name*[string]*)
- ProxyUsername(username*[string]*)
- ReadCSV(filename*[string]*, delimiter (optional)*[string]*, comment (optional)*[string]*)
- Rename(oldname*[string]*, newname*[string]*)
- Size(filename*[string]*)
- ToAbsolute(filename*[string]*, base*[string]*)
- ToRelative(filename*[string]*, base*[string]*)
- Upload(filename*[string]*, url*[string]*, options (optional)*[object]*)

Member functions

- Close()
- FindLineContaining(contain*[string]*)
- FindLineStarting(start*[string]*)
- Flush()
- ReadAll()
- ReadArrayBuffer(length (optional)*[integer]*)
- ReadChar()
- ReadLine()
- ReadLongLine()
- Seek(offset*[integer]*, origin (optional)*[constant]*)
- Tell()
- Write(string*[Any valid javascript type]*)
- WriteArrayBuffer(buffer*[ArrayBuffer]*, length (optional)*[integer]*)
- Writeln(string*[Any valid javascript type]*)

File constants

Name	Description
File.APPEND	Flag to open file for appending
File.BINARY	Flag to open file in binary mode. This will have no effect on unix/linux but for windows if a file is opened for writing with binary mode \n will not be translated to \r\n (CRLF), it will be written as \n (LF)

File.READ	Flag to open file for reading
File.UTF8	Flag to open file for reading as UTF-8 encoding.
File.WRITE	Flag to open file for writing

Constants for Find types

Name	Description
File.DIRECTORY	Find directories
File.FILE	Find files

Constants for Seek types

Name	Description
File.CURRENT	Seek relative to current file position
File.END	Seek relative to end of the file
File.START	Seek relative to start of the file

File properties

Name	Type	Description
filename (read only)	string	Name of the file
mode (read only)	constant	Mode the file was opened with (File.READ, File.WRITE etc)

Detailed Description

The File class gives you simple functions to read and write text files. The following simple example shows how to read from the file "/data/test/file.txt" and print each line read to the dialogue box:

```
var f, line;
f = new File("/data/test/file.txt", File.READ);
while ( (line = f.ReadLine()) != undefined)
{
    Message(line);
}
f.Close();
```

The following simple example shows how to write the numbers 1 to 10 to the file "/data/test/file.txt":

```
var n, line;
f = new File("/data/test/file.txt", File.WRITE);
for (n=1; n<=10; n++)
{
    f.Writeln(n);
}
f.Close();
```

See the documentation below for more details.

Constructor

new File(filename*[string]*, mode*[constant]*)

Description

Create a new File object for reading and writing text files.

Arguments

- **filename** (string)

Filename of the file you want to read/write. If reading, the file must exist. If writing, the file will be overwritten (if it exists) if mode is File.WRITE, or if mode is File.APPEND it will be appended to if it exists, or created if it does not. When reading a file the filename can also be a URL (uniform resource locator) in which case the file will be read from the remote site. See File.Get() for more details on the format of the URL.

- **mode** (constant)

The mode to open the file with. Can be File.READ, File.WRITE or File.APPEND. For File.WRITE or File.APPEND it can also be ORed with File.BINARY if required. By default text is read and written as ASCII. To read/write text in utf-8 mode can also be ORed with File.UTF8 if required.

Returns

File object

Return type

File

Example

To create a new file object to read file "/data/test/file.txt"

```
var f = new File("/data/test/file.txt", File.READ);
```

Details of functions

Close()

Description

Close a file opened by a File object.

Arguments

No arguments

Returns

No return value

Example

To close File object f.

```
f.Close();
```

Copy(source*[string]*, dest*[string]*) [static]

Description

Copies a file

Arguments

- **source** (string)

Source filename you want to copy.

- **dest** (string)

Destination filename you want to copy source file to.

Returns

true if copy successful, false otherwise.

Return type

Boolean

Example

To copy the file "/data/test/file.key" to "/data/test/file.key_backup"

```
var copied = File.Copy("/data/test/file.key", "/data/test/file.key_backup");
```

Delete(filename*[string]*) [static]

Description

Deletes a file

Arguments

- **filename** (string)

Filename you want to delete.

Returns

true if successful, false if not.

Return type

Boolean

Example

To delete the file "/data/test/file.key"

```
var deleted = File.Delete("/data/test/file.key");
```

DriveMapFilename(filename*[string]*, format*[constant]*) [static]

Description

Changes a filename or directory name to the correct format for a specific operating system using the directory mappings (if present)

Arguments

- **filename** (string)

Filename you want to drive map.

- **format** (constant)

The format for the file/directory name. Can be Include.NATIVE, Include.UNIX or Include.WINDOWS

Returns

string containing drive mapped filename

Return type

String

Example

If PRIMER has drive S: mapped to "/data" (by using the primer*drive_s, this*drive_s, d3plot*drive_s or oasys*drive_s preference)

```
var mapped = File.DriveMapFilename("/data/test/file.key", Include.WINDOWS);
```

mapped will be "S:\test\file.key".

```
var mapped = File.DriveMapFilename("S:\\test\\file.key", Include.UNIX);
```

mapped will be "/data/test/file.key".

Exists(filename*[string]*) [static]

Description

Check if a file exists. See also File.IsDirectory() and See also File.IsFile().

Arguments

- **filename** (string)

Filename you want to check for existance.

Returns

true/false

Return type

Boolean

Example

To see if the file "/data/test/file.key" exists

```
if (File.Exists("/data/test/file.key")) { do something }
```

FindFiles(directory*[string]*, type (optional)*[constant]*) [static]

Description

Find any files and/or directories in a directory.

Arguments

- **directory** (string)

Directory to look for files/directories in.

- **type (optional)** (constant)

Type of things to find. Can be bitwise OR of File.FILE and File.DIRECTORY. If omitted only files will be returned.

Returns

Array of filenames/directories

Return type

Array

Example

To return the filenames in the directory /data/test:

```
var fileList = File.FindFiles("/data/test")
```

To return the directories in the directory /data/test:

```
var fileList = File.FindFiles("/data/test", File.DIRECTORY)
```

To return the files and directories in the directory /data/test:

```
var fileList = File.FindFiles("/data/test", File.FILE|File.DIRECTORY)
```

FindLineContaining(contain*[string]*)

Description

Reads a line from a file which contains **contain**, opened for reading by a File object. Although this is possible using core JavaScript functions this function should be significantly faster as most of the processing is done by PRIMER in C rather than in the JavaScript interpreter. To enable this function to be as fast as possible a maximum line length of 512 characters is used. If you expect a file to have lines longer than 512 characters then use ReadLongLine which allows lines of any length. If one argument is used then the line must contain that string. If more than one argument is used then lines which contain any of the arguments will be returned

Arguments

- **contain** (string)

String which matching lines must contain

This argument can be repeated if required
Alternatively a single array argument containing the multiple values can be given

Returns

string read from file or `undefined` if end of file

Return type

String

Example

Loop, reading lines from File object f which contain 'example'.

```
var line;
while ( (line = f.FindLineContaining("example") ) != undefined)
{
}
```

FindLineStarting(start*[string]*)

Description

Reads a line from a file which starts with start, opened for reading by a File object. Although this is possible using core JavaScript functions this function should be significantly faster as most of the processing is done by PRIMER in C rather than in the JavaScript interpreter. To enable this function to be as fast as possible a maximum line length of 512 characters is used. If you expect a file to have lines longer than 512 characters then use ReadLongLine which allows lines of any length. If one argument is used then the line must start with that string. If more than one argument is used then lines which start with any of the arguments will be returned

Arguments

- **start** (string)

String which matching lines must start with

This argument can be repeated if required
Alternatively a single array argument containing the multiple values can be given

Returns

string read from file or `undefined` if end of file

Return type

String

Example

Loop, reading lines from File object f which start 'example'.

```
var line;
while ( (line = f.FindLineStarting("example") ) != undefined)
{
}
```

Flush()

Description

Flushes a file opened for writing by a File object.

Arguments

No arguments

Returns

No return value

Example

To flush File object f.

```
f.Flush();
```

Get(url*[string]*, filename*[string]*, options (optional)*[object]*) [static]

Description

Get a file from a remote location. See also File.Proxy(), File.ProxyPassword() and File.ProxyUsername().

Arguments

- **url** (string)

URL (uniform resource locator) of remote file you want to get. Currently http and ftp are supported. For http give the full address including the leading 'http://'. e.g.
'http://www.example.com/file.html'.
For ftp an optional username and password can be given. e.g.
'ftp://ftp.example.com' retrieves the directory listing for the root directory.
'ftp://ftp.example.com/readme.txt' downloads the file readme.txt from the root directory.
'ftp://user:password@ftp.example.com/readme.txt' retrieves the readme.txt file from the user's home directory.

- **filename** (string)

Filename you want to save the file to.

- **options (optional)** (object)

Options for get. If 'username' and 'password' are set then basic authorization using the username and password will be used.

Object has the following properties:

Name	Type	Description

password (optional)	string	Password
response (optional)	boolean	If set to true, then the response code will be returned instead of true/false. This can be used to retieve error messages and codes when the file is not returned successfully.
username (optional)	string	Username

Returns

true if file was successfully got, false otherwise.

Return type

Boolean

Example

To get the file "http://www.example.com/file.html" and save it to C:\temp:

```
File.Get("http://www.example.com/file.html", "C:\temp\file.html");
```

IsAbsolute(filename*[string]*) [static]

Description

Check if a filename is absolute or relative.

Arguments

- **filename** (string)

Filename you want to check.

Returns

true/false

Return type

Boolean

Example

To see if the filename "/data/test" is absolute (which it is!)

```
if (File.IsAbsolute("/data/test")) { do something }
```

IsDirectory(filename*[string]*) [static]

Description

Check if a filename is a directory. See also File.Exists(), File.IsFile(), File.IsReadable() and File.IsWritable().

Arguments

- **filename** (string)

Filename you want to check.

Returns

true/false

Return type

Boolean

Example

To see if the filename "/data/test" is a directory

```
if (File.IsDirectory("/data/test")) { do something }
```

IsFile(filename*[string]*) [static]

Description

Check if a filename is a file. See also File.Exists(), File.IsDirectory(), File.IsReadable() and File.IsWritable().

Arguments

- **filename** (string)

Filename you want to check.

Returns

true/false

Return type

Boolean

Example

To see if the filename "/data/test" is a file

```
if (File.IsFile("/data/test")) { do something }
```

IsReadable(filename*[string]*) [static]

Description

Check if a filename has read permissions. See also File.Exists(), File.IsDirectory() and File.IsWritable().

Arguments

- **filename** (string)

Filename you want to check.

Returns

true/false

Return type

Boolean

Example

To see if the filename "/data/test" is readable

```
if (File.IsReadable("/data/test")) { do something }
```

IsWritable(filename*[string]*) [static]

Description

Check if a filename has write permissions. If *filename* exists and it is a file then it is checked to see if it can be opened with write (File.APPEND permissions). If *filename* exists and it is a directory then the directory is checked for write permission (can files be created in the directory). If *filename* does not exist then it is assumed to be a file and is checked to see if it can be opened for writing (File.WRITE permissions). See also File.Exists(), File.IsDirectory() and File.IsReadable().

Arguments

- **filename** (string)

Filename you want to check.

Returns

true/false

Return type

Boolean

Example

To see if the filename "/data/test" is writable

```
if (File.IsWritable("/data/test")) { do something }
```

Mkdir(directory*[string]*) [static]

Description

Make a directory. If PRIMER preference 'directory_permission' is set e.g.755 then this will apply (same as if set by chmod 755) ignoring any setting of umask. If there is no preference then the users current setting of umask will control permissions (same as system mkdir)

Arguments

- **directory** (string)

The name of the directory you want to create.

Returns

true if successfully created, false if not.

Return type

Boolean

Example

To make the directory "/data/test"

```
var success = File.Mkdir("/data/test");
```

Mktemp() [static]

Description

Make a temporary filename for writing a temporary file.

Arguments

No arguments

Returns

String name of temporary filename that can be used.

Return type

String

Example

To get a temp filename"

```
var filename = File.Mktemp();
```

Proxy(name*[string]*) [static]

Description

Set a proxy for files opened by http, ftp etc. See also File.Get(), File.ProxyPassword() and File.ProxyUsername().

Arguments

- **name** (string)

The name of the proxy.

Returns

No return value

Example

To set the proxy to "http://example.proxy.com" using port 80:

```
File.Proxy("http://example.proxy.com:80");
```

ProxyPassword(name*[string]*) [static]

Description

Set a proxy password for files opened by http, ftp etc. See also File.Get(), File.Proxy() and File.ProxyUsername().

Arguments

- **name** (string)

Password for the proxy server.

Returns

No return value

Example

To set the proxy password to "password":

```
File.ProxyPassword("password");
```

ProxyUsername(username*[string]*) [static]

Description

Set a proxy username for files opened by http, ftp etc. See also File.Get(), File.Proxy() and File.ProxyPassword().

Arguments

- **username** (string)

The username for the proxy.

Returns

No return value

Example

To set the proxy username to "username":

```
File.ProxyUsername("username");
```

ReadAll()

Description

Reads **all** the remaining characters from a file opened for reading by a File object. As this function can read the entire file as a string be careful when reading large files as it will consume large amounts of memory.

Arguments

No arguments

Returns

String. Characters read from file or undefined if end of file

Return type

String

Example

Read all characters from File object f.

```
var c = f.ReadAll();
```

ReadArrayBuffer(length (optional)*[integer]*)

Description

Reads binary data from a file opened for reading by a File object. The data is returned as an ArrayBuffer object. For more details on how to use an ArrayBuffer see the following links:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Typed_arrays
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/ArrayBuffer
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/TypedArray
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/DataView.

Arguments

- **length (optional)** (integer)

Number of bytes to try to read from the file. If omitted all the remaining data from the file will be read.

Returns

ArrayBuffer object or undefined if end of file

Return type

ArrayBuffer

Example

To read data as 32bit unsigned integers from File object f.

```
var ab = f.ReadArrayBuffer();
var u32 = new Uint32Array(ab);
for (var i=0; i<u32.length; i++
{
    var value = u32[i];
}
```

ReadCSV(filename*[string]*, delimiter (optional)*[string]*, comment (optional)*[string]*) [static]

Description

Reads the input CSV file and returns an array of string arrays. If the CSV file has legitimate records the function returns an Array object containing sub-arrays of strings otherwise the function returns NULL. The lengths of all the sub-arrays are the same and equal to maximum number of fields in any of the records. For records in a CSV file having fewer fields, the respective sub-arrays are padded with NULL elements to the maximum array length.

Arguments

- **filename** (string)

Filename you want to read CSV options from.

- **delimiter (optional)** (string)

Delimiter string to be used. Default is a comma (",").

- **comment (optional)** (string)

Comment string to be used. Default is a dollar sign ("$").

Returns

2d array of strings.

Return type

String

Example

To Read CSV file "sample.csv" and print all records to a Window.

```
var csv_file_path = "C:\\sample.csv";
var records = "";
if(!File.Exists(csv_file_path))
{
    Window.Information("CSV file %s not present", csv_file_path);
    Exit();
}
var csv_array = File.ReadCSV(csv_file_path);
if(csv_array != null)
{
    for(var i = 0; i < csv_array.length; i++)
    {
        var record_array = csv_array[i];
        for(var j = 0; j < record_array.length; j++)
        {
            if(record_array[j] != null)
                records = records + record_array[j] + " , ";
        }
        records = records + "\n";
    }
}
Options.max_window_lines = csv_array.length;
Window.Information("File.ReadCSV Ouptut", records);
```

To Read CSV file "sample.csv" with delimiter string "::" and comment string "##".

```
var csv_array = File.ReadCSV(csv_file_path, "::", "##");
```

ReadChar()

Description

Reads a single character from a file opened for reading by a File object.

Arguments

No arguments

Returns

character read from file or

`undefined`

if end of file

Return type

String

Example

Loop, reading characters from File object f.

```
var c;
while ( (c = f.ReadChar()) != undefined) { ... }
```

ReadLine()

Description

Reads a line from a file opened for reading by a File object. To enable this function to be as fast as possible a maximum line length of 512 characters is used. If you expect a file to have lines longer than 512 characters then use ReadLongLine which allows lines of any length.

Arguments

No arguments

Returns

string read from file or

`undefined`

if end of file

Return type

String

Example

Loop, reading lines from File object f.

```
var line;
while ( (line = f.ReadLine()) != undefined) { ... }
```

ReadLongLine()

Description

Reads a line from a file opened for reading by a File object. The line can be any length. If your file has lines shorter than 512 characters then you may want to use ReadLine instead which is faster.

Arguments

No arguments

Returns

string read from file or

`undefined`

if end of file

Return type

String

Example

Loop, reading lines from File object f.

```
var line;
while ( (line = f.ReadLongLine()) != undefined) { ... }
```

Rename(oldname*[string]*, newname*[string]*) [static]

Description

Rename an existing file to have a different name.

Arguments

- **oldname** (string)

Existing filename you want to rename

- **newname** (string)

New filename you want to rename to

Returns

true if successful, false if not.

Return type

Boolean

Example

To rename the file "/data/test/file.key" to "/data/test/new_file.key"

```
var size = File.Rename("/data/test/file.key", "/data/test/new_file.key");
```

Seek(offset*[integer]*, origin (optional)*[constant]*)

Description

Set the current position for reading or writing in a File object.

Arguments

- **offset** (integer)

Offset to seek to in the file

- **origin (optional)** (constant)

Origin for offset. Must be one of File.START, File.END or File.CURRENT. If omitted File.START will be used.

Returns

no return value

Example

To seek to the end of File f:

```
f.Seek(0, File.END);
```

To seek to the beginning of File f:

```
f.Seek(0, File.START);
```

To move forward 10 characters in File f:

```
f.Seek(10, File.CURRENT);
```

Size(filename*[string]*) [static]

Description

Return the size of a file in bytes

Arguments

- **filename** (string)

Filename you want the size of.

Returns

size in bytes

Return type

Number

Example

To get the size of the file "/data/test/file.key"

```
var size = File.Size("/data/test/file.key");
```

Tell()

Description

Return the current file position for a File object. Note that on Windows when reading files if the file is not opened with File.BINARY this may not return the correct file position for files with unix line endings.

Arguments

No arguments

Returns

integer

Return type

Number

Example

To get the current file position for File f:

```
var pos = f.Tell();
```

ToAbsolute(filename*[string]*, base*[string]*) [static]

Description

Converts file paths to absolute file paths

Arguments

- **filename** (string)

File path you want to convert.

- **base** (string)

(Optional) Base file path. If omitted base will be the current working directory.

Returns

absolute file path

Return type

absolute file path

Example

To get the absolute file path for "\data\test\file.key" with base file path "C:\test\"

```
var absolute = File.ToAbsolute("data\\test\\file.key","C:\\test");
```

ToRelative(filename*[string]*, base*[string]*) [static]

Description

Converts file paths to relative file paths

Arguments

- **filename** (string)

File path you want to convert.

- **base** (string)

(Optional) Base file path. If omitted base will be the current working directory.

Returns

relative file path

Return type

relative file path

Example

To get the relative file path for "C:\test\data\test\file.key" with base file path "C:\test\"

```
var absolute = File.ToAbsolute("C:\\testdata\\test\\file.key","C:\\test");
```

Upload(filename*[string]*, url*[string]*, options (optional)*[object]*) [static]

Description

Uploads a file to a remote location. See also File.Proxy(), File.ProxyPassword() and File.ProxyUsername().

Arguments

- **filename** (string)

Filename you want to upload.

- **url** (string)

URL (uniform resource locator) of the remote location you want to upload the file to. Currently only http is supported. Give the full address including the leading 'http://'. e.g. 'http://www.example.com/file.html'.

- **options (optional)** (object)

Options for upload. If both of these are set then basic authorization using the username and password will be used.

Object has the following properties:

Name	Type	Description
password (optional)	string	Password
username (optional)	string	Username

Returns

true if file was successfully uploaded, false otherwise.

Return type

Boolean

Example

To upload the file "C:\temp\file.txt" to "http://www.example.com/file.txt":

```
File.Upload("C:/temp/file.txt", "http://www.example.com/file.txt");
```

Write(string*[Any valid javascript type]*)

Description

Write a string to a file opened for writing by a File object. **Note that a carriage return is not added**.

Arguments

- **string** (Any valid javascript type)

The string/item that you want to write

Returns

No return value

Example

To write string "Hello, world!" to File object f

```
f.Write("Hello, world!\n");
```

To write the title of model m to File object f

```
f.Write("The title of model 2 is " + m.title + "\n");
```

WriteArrayBuffer(buffer*[ArrayBuffer]*, length (optional)*[integer]*)

Description

Writes binary data to a file opened for writing by a File object. The data to write is an ArrayBuffer object. For more details on how to use an ArrayBuffer see the following links:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Typed_arrays
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/ArrayBuffer
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/TypedArray
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/DataView.

Arguments

- **buffer** (ArrayBuffer)

ArrayBuffer to write to file

- **length (optional)** (integer)

Number of bytes to write to the file. If omitted all the data in the ArrayBuffer will be written (buffer.byteLength bytes)

Returns

No return value

Example

To write ArrayBuffer ab to File object f.

```
f.WriteArrayBuffer(ab);
```

Writeln(string*[Any valid javascript type]*)

Description

Write a string to a file opened for writing by a File object **adding a carriage return**.

Arguments

- **string** (Any valid javascript type)

The string/item that you want to write

Returns

No return value

Example

To write string "Hello, world!" to File object f automatically adding a carriage return

```
f.Writeln("Hello, world!");
```

To write the title of model m to File object f automatically adding a carriage return

```
f.Writeln("The title of model 2 is " + m.title);
```

GeometrySurface class

The GeometrySurface class gives you access to surfaces in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- CalculateNormal(u*[real]*, y*[real]*)
- CalculatePoint(u*[real]*, v*[real]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetEdgeIndices()
- GetParameter(prop*[string]*)
- GetTriaIndices()
- GetVertices()
- Next()
- Previous()
- ProjectPoint(x*[real]*, y*[real]*, z*[real]*)
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)

- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()

GeometrySurface properties

Name	Type	Description
exists (read only)	logical	true if surface exists, false if referred to but not defined.
id (read only)	integer	GeometrySurface number. Also see the label property which is an alternative name for this.
include	integer	The Include file number that the surface is in.
label (read only)	integer	GeometrySurface number. Also see the id property which is an alternative name for this.
model (read only)	integer	The Model number that the surface is in.

Detailed Description

The GeometrySurface class allows you to create, modify, edit and manipulate surfaces cards. See the documentation below for more details.

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a surface.

Arguments

- **Comment** (Comment)

Comment that will be attached to the surface

Returns

No return value

Example

To associate comment c to the surface s:

```
s.AssociateComment(c);
```

Blank()

Description

Blanks the surface

Arguments

No arguments

Returns

No return value

Example

To blank surface s:

```
s.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the surfaces in the model.

Arguments

- **Model** (Model)

Model that all surfaces will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the surfaces in model m:

```
GeometrySurface.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged surfaces in the model.

Arguments

- **Model** (Model)

Model that all the flagged surfaces will be blanked in

- **flag** (Flag)

Flag set on the surfaces that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the surfaces in model m flagged with f:

```
GeometrySurface.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the surface is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if surface s is blanked:

```
if (s.Blanked() ) do_something...
```

CalculateNormal(u*[real]*, y*[real]*)

Description

Calculate the normal vector for a parametric point on a surface.

Arguments

- **u** (real)

u parametric coordinate

- **y** (real)

v parametric coordinate

Returns

Array containing x, y and z values.

Return type

Array

Example

To obtain the surface normal at parametric point (0.2, 0.3) on surface s:

```
var coords = s.CalculateNormal(0.2, 0.3);
```

CalculatePoint(u*[real]*, v*[real]*)

Description

Calculate the X, Y and Z coordinates for a parametric point on a surface.

Arguments

- **u** (real)

u parametric coordinate

- **v** (real)

v parametric coordinate

Returns

Array containing x, y and z values.

Return type

Array

Example

To obtain the coordinates of parametric point (0.2, 0.3) on surface s:

```
var coords = s.CalculatePoint(0.2, 0.3);
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the surface.

Arguments

- **flag** (Flag)

Flag to clear on the surface

Returns

No return value

Example

To clear flag f for surface s:

```
s.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the surface. The target include of the copied surface can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

GeometrySurface object

Return type

GeometrySurface

Example

To copy surface s into surface z:

```
var z = s.Copy();
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a surface.

Arguments

- **Comment** (Comment)

Comment that will be detached from the surface

Returns

No return value

Example

To detach comment c from the surface s:

```
s.DetachComment(c);
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for surface with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The surface object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for surface s:

```
s.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for surface s:

```
s.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for surface. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for surface s:

```
s.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first surface in the model.

Arguments

- **Model** (Model)

Model to get first surface in

Returns

GeometrySurface object (or null if there are no surfaces in the model).

Return type

GeometrySurface

Example

To get the first surface in model m:

```
var s = GeometrySurface.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free surface label in the model. Also see GeometrySurface.LastFreeLabel(), GeometrySurface.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free surface label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

GeometrySurface label.

Return type

Number

Example

To get the first free surface label in model m:

```
var label = GeometrySurface.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the surfaces in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all surfaces will be flagged in

- **flag** (Flag)

Flag to set on the surfaces

Returns

No return value

Example

To flag all of the surfaces with flag f in model m:

```
GeometrySurface.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the surface is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the surface

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if surface s has flag f set on it:

```
if (s.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each surface in the model.
Note that ForEach has been designed to make looping over surfaces as fast as possible and so has some limitations.
Firstly, a single temporary GeometrySurface object is created and on each function call it is updated with the current surface data. This means that you should not try to store the GeometrySurface object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new surfaces inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all surfaces are in

- **func** (function)

Function to call for each surface

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the surfaces in model m:

```
GeometrySurface.ForEach(m, test);
function test(s)
{
// s is GeometrySurface object
}
```

To call function test for all of the surfaces in model m with optional object:

```
var data = { x:0, y:0 };
GeometrySurface.ForEach(m, test, data);
function test(s, extra)
{
// s is GeometrySurface object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of GeometrySurface objects or properties for all of the surfaces in a model in PRIMER. If the optional property argument is not given then an array of GeometrySurface objects is returned. If the property argument is given, that property value for each surface is returned in the array instead of a GeometrySurface object

Arguments

- **Model** (Model)

Model to get surfaces from

- **property (optional)** (string)

Name for property to get for all surfaces in the model

Returns

Array of GeometrySurface objects or properties

Return type

Array

Example

To make an array of GeometrySurface objects for all of the surfaces in model m:

```
var a = GeometrySurface.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each surface in model m:

```
var a = GeometrySurface.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a surface.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the surface s:

```
var comm_array = s.GetComments();
```

GetEdgeIndices()

Description

Return an array of all the edge indices for a surface (in pairs).

Arguments

No arguments

Returns

Array of indices

Return type

Array

Example

To get edge indices for surface s

```
var edges = s.GetEdgeIndices();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of GeometrySurface objects for all of the flagged surfaces in a model in PRIMER If the optional property argument is not given then an array of GeometrySurface objects is returned. If the property argument is given, then that property value for each surface is returned in the array instead of a GeometrySurface object

Arguments

- **Model** (Model)

Model to get surfaces from

- **flag** (Flag)

Flag set on the surfaces that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged surfaces in the model

Returns

Array of GeometrySurface objects or properties

Return type

Array

Example

To make an array of GeometrySurface objects for all of the surfaces in model m flagged with f:

```
var s = GeometrySurface.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the surfaces in model m flagged with f:

```
var a = GeometrySurface.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the GeometrySurface object for a surface ID.

Arguments

- **Model** (Model)

Model to find the surface in

- **number** (integer)

number of the surface you want the GeometrySurface object for

Returns

GeometrySurface object (or null if surface does not exist).

Return type

GeometrySurface

Example

To get the GeometrySurface object for surface 100 in model m

```
var s = GeometrySurface.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a GeometrySurface property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the GeometrySurface.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

surface property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if GeometrySurface property s.example is a parameter:

```
Options.property_parameter_names = true;
if (s.GetParameter(s.example) ) do_something...
Options.property_parameter_names = false;
```

To check if GeometrySurface property s.example is a parameter by using the GetParameter method:

```
if (s.ViewParameters().GetParameter(s.example) ) do_something...
```

GetTriaIndices()

Description

Return an array of all the tria indices for a surface (in triplets).

Arguments

No arguments

Returns

Array of indices

Return type

Array

Example

To get tria indices for surface s

```
var trias = s.GetTriaIndices();
```

GetVertices()

Description

Return an array of all the vertex coordinates for a surface (in triplets).

Arguments

No arguments

Returns

Array of indices

Return type

Array

Example

To get vertex coordinates for surface s

```
var vertices = s.GetVertices();
```

Last(Model*[Model]*) [static]

Description

Returns the last surface in the model.

Arguments

- **Model** (Model)

Model to get last surface in

Returns

GeometrySurface object (or null if there are no surfaces in the model).

Return type

GeometrySurface

Example

To get the last surface in model m:

```
var s = GeometrySurface.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free surface label in the model. Also see GeometrySurface.FirstFreeLabel(), GeometrySurface.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free surface label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

GeometrySurface label.

Return type

Number

Example

To get the last free surface label in model m:

```
var label = GeometrySurface.LastFreeLabel(m);
```

Next()

Description

Returns the next surface in the model.

Arguments

No arguments

Returns

GeometrySurface object (or null if there are no more surfaces in the model).

Return type

GeometrySurface

Example

To get the surface in model m after surface s:

```
var s = s.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) surface label in the model. Also see GeometrySurface.FirstFreeLabel(), GeometrySurface.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free surface label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

GeometrySurface label.

Return type

Number

Example

To get the next free surface label in model m:

```
var label = GeometrySurface.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a surface.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only surfaces from that model can be picked. If the argument is a Flag then only surfaces that are flagged with *limit* can be selected. If omitted, or null, any surfaces from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

GeometrySurface object (or null if not picked)

Return type

GeometrySurface

Example

To pick a surface from model m giving the prompt 'Pick surface from screen':

```
var s = GeometrySurface.Pick('Pick surface from screen', m);
```

Previous()

Description

Returns the previous surface in the model.

Arguments

No arguments

Returns

GeometrySurface object (or null if there are no more surfaces in the model).

Return type

GeometrySurface

Example

To get the surface in model m before surface s:

```
var s = s.Previous();
```

ProjectPoint(x*[real]*, y*[real]*, z*[real]*)

Description

Project a point onto the surface.

Arguments

- **x** (real)

X coordinate of point to project

- **y** (real)

Y coordinate of point to project

- **z** (real)

Z coordinate of point to project

Returns

Array containing u and v values.

Return type

Array

Example

To obtain the projection of point (1, 2, 3) on to surface s:

```
var projection = s.ProjectPoint(1, 2, 3);
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the surfaces in the model.

Arguments

- **Model** (Model)

Model that all surfaces will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the surfaces in model m, from 1000000:

```
GeometrySurface.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged surfaces in the model.

Arguments

- **Model** (Model)

Model that all the flagged surfaces will be renumbered in

- **flag** (Flag)

Flag set on the surfaces that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the surfaces in model m flagged with f, from 1000000:

```
GeometrySurface.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select surfaces using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting surfaces

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only surfaces from that model can be selected. If the argument is a Flag then only surfaces that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any surfaces can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of surfaces selected or null if menu cancelled

Return type

Number

Example

To select surfaces from model m, flagging those selected with flag f, giving the prompt 'Select surfaces':

```
GeometrySurface.Select(f, 'Select surfaces', m);
```

To select surfaces, flagging those selected with flag f but limiting selection to surfaces flagged with flag l, giving the prompt 'Select surfaces':

```
GeometrySurface.Select(f, 'Select surfaces', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the surface.

Arguments

- **flag** (Flag)

Flag to set on the surface

Returns

No return value

Example

To set flag f for surface s:

```
s.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the surface. The surface will be sketched until you either call GeometrySurface.Unsketch(), GeometrySurface.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the surface is sketched. If omitted redraw is true. If you want to sketch several surfaces and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch surface s:

```
s.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged surfaces in the model. The surfaces will be sketched until you either call GeometrySurface.Unsketch(), GeometrySurface.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged surfaces will be sketched in

- **flag** (Flag)

Flag set on the surfaces that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the surfaces are sketched. If omitted redraw is true. If you want to sketch flagged surfaces several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all surfaces flagged with flag in model m:

```
GeometrySurface.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of surfaces in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing surfaces should be counted. If false or omitted referenced but undefined surfaces will also be included in the total.

Returns

number of surfaces

Return type

Number

Example

To get the total number of surfaces in model m:

```
var total = GeometrySurface.Total(m);
```

Unblank()

Description

Unblanks the surface

Arguments

No arguments

Returns

No return value

Example

To unblank surface s:

```
s.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the surfaces in the model.

Arguments

- **Model** (Model)

Model that all surfaces will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the surfaces in model m:

```
GeometrySurface.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged surfaces in the model.

Arguments

- **Model** (Model)

Model that the flagged surfaces will be unblanked in

- **flag** (Flag)

Flag set on the surfaces that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the surfaces in model m flagged with f:

```
GeometrySurface.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the surfaces in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all surfaces will be unset in

- **flag** (Flag)

Flag to unset on the surfaces

Returns

No return value

Example

To unset the flag f on all the surfaces in model m:

```
GeometrySurface.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the surface.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the surface is unsketched. If omitted redraw is true. If you want to unsketch several surfaces and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch surface s:

```
s.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all surfaces.

Arguments

- **Model** (Model)

Model that all surfaces will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the surfaces are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all surfaces in model m:

```
GeometrySurface.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged surfaces in the model.

Arguments

- **Model** (Model)

Model that all surfaces will be unsketched in

- **flag** (Flag)

Flag set on the surfaces that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the surfaces are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all surfaces flagged with flag in model m:

```
GeometrySurface.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

GeometrySurface object.

Return type

GeometrySurface

Example

To check if GeometrySurface property s.example is a parameter by using the GeometrySurface.GetParameter() method:

```
if (s.ViewParameters().GetParameter(s.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for surface with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description

autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The surface object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for surface s:

```
s.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for surface s:

```
s.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for surface. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for surface s:

```
s.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this surface.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for surface s:

```
var xrefs = s.Xrefs();
```

Graphics class

The Graphics class allows you to draw graphics in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- DepthTest(enable*[boolean]*)
- DrawingFunction(name*[function]*)
- FillColour(colour*[Colour]*)
- Finish()
- Line(x1*[real]*, y1*[real]*, z1*[real]*, x2*[real]*, y2*[real]*, z2*[real]*)
- LineColour(colour*[Colour]*)
- LineStyle(style*[constant]*)
- LineTo(x*[real]*, y*[real]*, z*[real]*)
- LineWidth(width*[Integer]*)
- MoveTo(x*[real]*, y*[real]*, z*[real]*)
- PolygonFinish()
- PolygonStart()
- Shape(shape*[constant]*, size*[integer]*)
- Start()
- Text(text*[String]*)
- TextColour(colour*[Colour]*)
- TextSize(size*[Integer]*)

Graphics constants

Name	Description
Graphics.CIRCLE	Circle shape. See Graphics.Shape() for use.
Graphics.DASHDOT_LINE	Dashed and dotted lines. See Graphics.LineStyle() for use.
Graphics.DASH_LINE	Dashed lines. See Graphics.LineStyle() for use.
Graphics.DIAMOND	Diamond shape. See Graphics.Shape() for use.
Graphics.DOT_LINE	Dotted lines. See Graphics.LineStyle() for use.
Graphics.FILLED_CIRCLE	Filled circle shape. See Graphics.Shape() for use.
Graphics.FILLED_DIAMOND	Filled diamond shape. See Graphics.Shape() for use.
Graphics.FILLED_HOURGLASS	Filled hourglass shape. See Graphics.Shape() for use.
Graphics.FILLED_SQUARE	Filled square shape. See Graphics.Shape() for use.
Graphics.HOURGLASS	Hourglass shape. See Graphics.Shape() for use.
Graphics.POINT	Point shape. See Graphics.Shape() for use.
Graphics.SOLID_LINE	Solid lines. See Graphics.LineStyle() for use.
Graphics.SQUARE	Square shape. See Graphics.Shape() for use.

Detailed Description

The Graphics class gives you access to functions to draw lines, shapes etc on the graphics screen in PRIMER. For example the following will draw a solid thick red line on the screen:

```
Graphics.Start();
Graphics.LineWidth(3);
Graphics.LineColour(Colour.RED);
Graphics.LineStyle(Graphics.SOLID_LINE);
Graphics.Line(0, 0, 0, 10, 20, 30);
Graphics.Finish();
```

The drawing commands must be between

```
Graphics.Start()
```

and

```
Graphics.Finish()
```

or else nothing will be seen. This is suitable for sketching but the line will disappear if the graphics are redrawn or any dynamic viewing is done. To draw graphics which will stay on the screen even if dynamic viewing or a redraw is done you have to register a function using Graphics.DrawingFunction() which will be called every time the graphics are redrawn by PRIMER. e.g:

```
var w = new Window("Graphics test", 0.8, 1.0, 0.5, 0.6);
var e = new Widget(w, Widget.BUTTON,  1, 21, 1, 7, "Exit");
e.onClick  = Exit;
do_draw();
Graphics.DrawingFunction(do_draw);
w.Show();
////////////////////////////////////////////////////////////////////////////
function do_draw()
{
    Graphics.Start();
    Graphics.LineWidth(3);
    Graphics.LineColour(Colour.RED);
    Graphics.LineStyle(Graphics.SOLID_LINE);
    Graphics.Line(0, 0, 0, 10, 20, 30);
    Graphics.Finish();
}
```

See the documentation below for more details.

Details of functions

DepthTest(enable*[boolean]*) [static]

Description

Allows depth testing (hidden surface removal) to be turned on or off. Temporarily turning depth testing off may be used to ensure that an item (e.g. some text) is always drawn in front and will not be obscured.

Arguments

- **enable** (boolean)

Whether depth testing (hidden surface removal) is performed (true) or not (false)

Returns

No return value

Example

To turn off depth testing:

```
Graphics.DepthTest(false);
```

To turn depth testing back on:

```
Graphics.DepthTest(true);
```

DrawingFunction(name*[function]*) [static]

Description

Set the function to draw graphics from javaScript. This function will be called each time the graphics are redrawn after PRIMER has finished drawing everything else. This allows you to add extra items to the graphics.
To remove the graphics drawing function use Graphics.DrawingFunction(null).
It is the responsibility of the script developer to ensure that any objects or variables that are used in the drawing function do not refer to items in PRIMER that no longer exist. Not doing so may cause PRIMER to crash. For example, if you use some Node objects in the drawing function that refer to nodes in model 1 and you delete the model, when the graphics are redrawn PRIMER may crash as the nodes referred to by the Node objects no longer exist. You should either remove the drawing function by calling Graphics.DrawingFunction(null) or set the Node variables to null (and test that they exist before using them) in your drawing function **before** deleting the model.
If a drawing function is used in your script, you should reset it before the script terminates to avoid a "race condition" between the script terminating and the graphics function being called. Not doing so may cause PRIMER to crash.

Arguments

- **name** (function)

The name of the function (or null to remove a function)

Returns

No return value

Example

To set function MyRedrawFunction as the Graphics drawing function

```
Graphics.DrawingFunction(MyRedrawFunction);
```

FillColour(colour*[Colour]*) [static]

Description

Sets the colour for drawing polygons. See the Colour class for more details on colours.

Arguments

- **colour** (Colour)

The colour you want to fill polygons with

Returns

No return value

Example

To Set the current fill colour to red:

```
Graphics.FillColour(Colour.RED);
```

or

```
Graphics.FillColour( Colour.RGB(255, 0, 0) );
```

Finish() [static]

Description

Finish any graphics. See also Graphics.Start(). This **must** be used to finish drawing.

Arguments

No arguments

Returns

No return value

Example

To finish any graphics operations:

```
Graphics.Finish();
```

Line(x1*[real]*, y1*[real]*, z1*[real]*, x2*[real]*, y2*[real]*, z2*[real]*) [static]

Description

Draws a line from (x1, y1, z1) to (x2, y2, z2). See also Graphics.LineTo() and Graphics.MoveTo()

Arguments

- **x1** (real)

X coordinate of point 1

- **y1** (real)

Y coordinate of point 1

- **z1** (real)

Z coordinate of point 1

- **x2** (real)

X coordinate of point 2

- **y2** (real)

Y coordinate of point 2

- **z2** (real)

Z coordinate of point 2

Returns

No return value

Example

To draw a line from (0.0, 0.0, 0.0) to (10.0, 20.0, 30.0)

```
Graphics.Line(0.0, 0.0, 0.0, 10.0, 20.0, 30.0);
```

LineColour(colour*[Colour]*) [static]

Description

Sets the colour for drawing lines. See the Colour class for more details on colours.

Arguments

- **colour** (Colour)

The colour you want to draw lines with

Returns

No return value

Example

To Set the current drawing colour to red:

```
Graphics.LineColour(Colour.RED);
```

or

```
Graphics.LineColour( Colour.RGB(255, 0, 0) );
```

LineStyle(style*[constant]*) [static]

Description

Sets the style for drawing lines.

Arguments

- **style** (constant)

The style to draw lines with. Can be: Graphics.SOLID_LINE, Graphics.DASH_LINE, Graphics.DASHDOT_LINE or Graphics.DOT_LINE

Returns

No return value

Example

To Set the current line style to 3:

```
Graphics.LineStyle(3);
```

LineTo(x*[real]*, y*[real]*, z*[real]*) [static]

Description

Draws a line from the current point to (x, y, z). After drawing the line the current point will be (x, y, z). See also Graphics.Line() and Graphics.MoveTo()

Arguments

- **x** (real)

X coordinate

- **y** (real)

Y coordinate

- **z** (real)

Z coordinate

Returns

No return value

Example

To draw a line from the current point to (10.0, 20.0, 30.0):

```
Graphics.LineTo(10.0, 20.0, 30.0);
```

LineWidth(width*[Integer]*) [static]

Description

Sets the width for drawing lines.

Arguments

- **width** (Integer)

The width to draw lines with

Returns

No return value

Example

To Set the current line width to 3:

```
Graphics.LineWidth(3);
```

MoveTo(x*[real]*, y*[real]*, z*[real]*) [static]

Description

Sets the current point to (x, y, z). See also Graphics.Line() and Graphics.LineTo()

Arguments

- **x** (real)

X coordinate

- **y** (real)

Y coordinate

- **z** (real)

Z coordinate

Returns

No return value

Example

To set the current point to (10.0, 20.0, 30.0):

```
Graphics.MoveTo(10.0, 20.0, 30.0);
```

PolygonFinish() [static]

Description

Ends drawing a polygon. See also Graphics.PolygonStart()

Arguments

No arguments

Returns

No return value

Example

To draw a red square:

```
Graphics.FillColour(Colour.RED);
Graphics.MoveTo(0.0, 0.0, 0.0);
Graphics.PolygonStart();
Graphics.MoveTo(10.0, 0.0, 0.0);
Graphics.MoveTo(10.0, 10.0, 0.0);
Graphics.MoveTo(0.0, 10.0, 0.0);
Graphics.PolygonFinish();
```

PolygonStart() [static]

Description

Starts drawing a **convex**polygon. A maximum of 250 vertices are allowed. Drawing concave polygons is not supported by this function. To draw concave polygons split them into separate convex polygons.
See also Graphics.PolygonFinish().
The only graphics command allowed between Graphics.PolygonStart() and Graphics.PolygonFinish() is Graphics.MoveTo(). Any other graphics drawing commands (e.g. Graphics.Text()) will be ignored.

Arguments

No arguments

Returns

No return value

Example

To draw a red square:

```
Graphics.FillColour(Colour.RED);
Graphics.MoveTo(0.0, 0.0, 0.0);
Graphics.PolygonStart();
Graphics.MoveTo(10.0, 0.0, 0.0);
Graphics.MoveTo(10.0, 10.0, 0.0);
Graphics.MoveTo(0.0, 10.0, 0.0);
Graphics.PolygonFinish();
```

Shape(shape*[constant]*, size*[integer]*) [static]

Description

Draws a simple shape.

Arguments

- **shape** (constant)

The style to draw lines with. Can be: Graphics.POINT, Graphics.SQUARE, Graphics.CIRCLE, Graphics.DIAMOND, Graphics.HOURGLASS, Graphics.FILLED_SQUARE, Graphics.FILLED_CIRCLE, Graphics.FILLED_DIAMOND or Graphics.FILLED_HOURGLASS

- **size** (integer)

Size the shape should be drawn at.

Returns

No return value

Example

To draw a filled square at (10, 20, 30) at size 10:

```
Graphics.MoveTo(10, 20, 30);
Graphics.Shape(Graphics.FILLED_SQUARE, 10);
```

Start() [static]

Description

Start any graphics. See also Graphics.Finish(). This **must** be used before any drawing is done.

Arguments

No arguments

Returns

No return value

Example

To start drawing graphics:

```
Graphics.Start();
```

Text(text*[String]*) [static]

Description

Draws text at current position. See Graphics.MoveTo() to set the current position.

Arguments

- **text** (String)

The text to write

Returns

No return value

Example

To write "Example" at (10, 20, 30):

```
Graphics.MoveTo(10, 20, 30);
Graphics.Text("Example");
```

TextColour(colour*[Colour]*) [static]

Description

Sets the colour for drawing text. See the Colour class for more details on colours.

Arguments

- **colour** (Colour)

The colour you want to draw text with

Returns

No return value

Example

To Set the current text drawing colour to red:

```
Graphics.TextColour(Colour.RED);
```

or

```
Graphics.TextColour( Colour.RGB(255, 0, 0) );
```

TextSize(size*[Integer]*) [static]

Description

Sets the size for drawing text.

Arguments

- **size** (Integer)

The size to draw text with

Returns

No return value

Example

To Set the current text size to 30:

```
Graphics.TextSize(30);
```

Group class

The Group class gives you access to groups in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Create(Model*[Model]*, modal (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Edit(modal (optional)*[boolean]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetDataAll(type*[string]*, index*[integer]*)
- GetDataList(type*[string]*, index*[integer]*)
- GetDataRange(type*[string]*, index*[integer]*)
- GetParameter(prop*[string]*)
- GetTotalAll(type*[string]*)
- GetTotalList(type*[string]*)
- GetTotalRange(type*[string]*)
- GetTotals(type*[string]*)
- GetType(row*[integer]*)
- Keyword()
- KeywordCards()
- Next()

- Previous()
- RemoveDataAll(type*[string]*, index*[integer]*)
- RemoveDataList(type*[string]*, index*[integer]*)
- RemoveDataRange(type*[string]*, index*[integer]*)
- SetDataAll(type*[string]*, index*[integer]*, data*[Array of data]*)
- SetDataList(type*[string]*, index*[integer]*, data*[Array of data]*)
- SetDataRange(type*[string]*, index*[integer]*, data*[Array of data]*)
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

Group constants

Name	Description
Group.ADD	Add contents to group
Group.REMOVE	Remove contents from group

Group properties

Name	Type	Description
exists (read only)	logical	true if group exists, false if referred to but not defined.
include	integer	The Include file number that the group is in.
label	integer	Group number
lock	logical	Whether Group contents are locked against deletion.
model (read only)	integer	The Model number that the group is in.
numtypes (read only)	integer	Number of types in the group.
title	string	Group title

Detailed Description

The Group class allows you to create, modify, edit and manipulate groups. See the documentation below for more details.

Constructor

new Group(Model*[Model]*, label*[integer]*, title (optional)*[string]*)

Description

Create a new Group object.

Arguments

- **Model** (Model)

Model that Group will be created in

- **label** (integer)

Group number.

- **title (optional)** (string)

Title for the group

Returns

Group object

Return type

Group

Example

To create a new group 99 in model m with title "Example":

```
var g = new Group(m, 99, "Example");
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a group.

Arguments

- **Comment** (Comment)

Comment that will be attached to the group

Returns

No return value

Example

To associate comment c to the group g:

```
g.AssociateComment(c);
```

Blank()

Description

Blanks the group

Arguments

No arguments

Returns

No return value

Example

To blank group g:

```
g.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the groups in the model.

Arguments

- **Model** (Model)

Model that all groups will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the groups in model m:

```
Group.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged groups in the model.

Arguments

- **Model** (Model)

Model that all the flagged groups will be blanked in

- **flag** (Flag)

Flag set on the groups that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the groups in model m flagged with f:

```
Group.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the group is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if group g is blanked:

```
if (g.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse group g:

```
g.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the group.

Arguments

- **flag** (Flag)

Flag to clear on the group

Returns

No return value

Example

To clear flag f for group g:

```
g.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the group. The target include of the copied group can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

Group object

Return type

Group

Example

To copy group g into group z:

```
var z = g.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a group

Arguments

- **Model** (Model)

Model that the group will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

Group object (or null if not made).

Return type

Group

Example

To start creating a group in model m:

```
var g = Group.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a group.

Arguments

- **Comment** (Comment)

Comment that will be detached from the group

Returns

No return value

Example

To detach comment c from the group g:

```
g.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit group g:

```
g.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for group with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The group object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for group g:

```
g.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for group g:

```
g.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for group. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for group g:

```
g.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first group in the model.

Arguments

- **Model** (Model)

Model to get first group in

Returns

Group object (or null if there are no groups in the model).

Return type

Group

Example

To get the first group in model m:

```
var g = Group.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free group label in the model. Also see Group.LastFreeLabel(), Group.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free group label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

Group label.

Return type

Number

Example

To get the first free group label in model m:

```
var label = Group.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the groups in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all groups will be flagged in

- **flag** (Flag)

Flag to set on the groups

Returns

No return value

Example

To flag all of the groups with flag f in model m:

```
Group.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the group is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the group

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if group g has flag f set on it:

```
if (g.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each group in the model.
Note that ForEach has been designed to make looping over groups as fast as possible and so has some limitations. Firstly, a single temporary Group object is created and on each function call it is updated with the current group data. This means that you should not try to store the Group object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new groups inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all groups are in

- **func** (function)

Function to call for each group

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the groups in model m:

```
Group.ForEach(m, test);
function test(g)
{
// g is Group object
}
```

To call function test for all of the groups in model m with optional object:

```
var data = { x:0, y:0 };
Group.ForEach(m, test, data);
function test(g, extra)
{
// g is Group object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Group objects or properties for all of the groups in a model in PRIMER. If the optional property argument is not given then an array of Group objects is returned. If the property argument is given, that property value for each group is returned in the array instead of a Group object

Arguments

- **Model** (Model)

Model to get groups from

- **property (optional)** (string)

Name for property to get for all groups in the model

Returns

Array of Group objects or properties

Return type

Array

Example

To make an array of Group objects for all of the groups in model m:

```
var a = Group.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each group in model m:

```
var a = Group.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a group.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the group g:

```
var comm_array = g.GetComments();
```

GetDataAll(type*[string]*, index*[integer]*)

Description

Returns 'all' data for a given row number and type in the group.

Arguments

- **type** (string)

The type of the item

- **index** (integer)

Index of 'all' row you want the data for. **Note that indices start at 0, not 1**.
0 <= index < Group.GetTotalAll()

Returns

An array containing data [Group.ADD or Group.REMOVE, BOX (if defined)].

Return type

Array

Example

To get the data for the 3rd SHELL 'all' row in group g:

```
var data = g.GetDataAll("SHELL", 2);
```

GetDataList(type*[string]*, index*[integer]*)

Description

Returns 'list' data for a given row number and type in the group.

Arguments

- **type** (string)

The type of the item

- **index** (integer)

Index of 'list' row you want the data for. **Note that indices start at 0, not 1**.
0 <= index < Group.GetTotalList()

Returns

An array containing data [Group.ADD or Group.REMOVE, ITEM1 (if defined), ITEM2 (if defined), ITEM3 (if defined), ITEM4 (if defined), ITEM5 (if defined), BOX (if defined)].

Return type

Array

Example

To get the data for the 3rd SHELL 'list' row in group g:

```
var data = g.GetDataList("SHELL", 2);
```

GetDataRange(type*[string]*, index*[integer]*)

Description

Returns 'range' data for a given row number and type in the group.

Arguments

- **type** (string)

The type of the item

- **index** (integer)

Index of 'range' row you want the data for. **Note that indices start at 0, not 1**.
0 <= index < Group.GetTotalRange()

Returns

An array containing data [Group.ADD or Group.REMOVE, START, END, BOX (if defined)].

Return type

Array

Example

To get the data for the 3rd SHELL 'range' row in group g:

```
var data = g.GetDataRange("SHELL", 2);
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of Group objects for all of the flagged groups in a model in PRIMER If the optional property argument is not given then an array of Group objects is returned. If the property argument is given, then that property value for each group is returned in the array instead of a Group object

Arguments

- **Model** (Model)

Model to get groups from

- **flag** (Flag)

Flag set on the groups that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged groups in the model

Returns

Array of Group objects or properties

Return type

Array

Example

To make an array of Group objects for all of the groups in model m flagged with f:

```
var g = Group.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the groups in model m flagged with f:

```
var a = Group.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the Group object for a group ID.

Arguments

- **Model** (Model)

Model to find the group in

- **number** (integer)

number of the group you want the Group object for

Returns

Group object (or null if group does not exist).

Return type

Group

Example

To get the Group object for group 100 in model m

```
var g = Group.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a Group property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the Group.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

group property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if Group property g.example is a parameter:

```
Options.property_parameter_names = true;
if (g.GetParameter(g.example) ) do_something...
Options.property_parameter_names = false;
```

To check if Group property g.example is a parameter by using the GetParameter method:

```
if (g.ViewParameters().GetParameter(g.example) ) do_something...
```

GetTotalAll(type*[string]*)

Description

Returns the total number of 'all' rows for a type in a group

Arguments

- **type** (string)

The type of the item

Returns

The number of ’all’ rows defined

Return type

Number

Example

To get the total number of shell ’all’ rows in group g:

```
var nrow = g.GetTotalAll("SHELL");
```

GetTotalList(type*[string]*)

Description

Returns the total number of ’list’ rows for a type in a group

Arguments

- **type** (string)

The type of the item

Returns

The number of ’list’ rows defined

Return type

Number

Example

To get the total number of shell ’list’ rows in group g:

```
var nrow = g.GetTotalList("SHELL");
```

GetTotalRange(type*[string]*)

Description

Returns the total number of ’range’ rows for a type in a group

Arguments

- **type** (string)

The type of the item

Returns

The number of ’range’ rows defined

Return type

Number

Example

To get the total number of shell ’range’ rows in group g:

```
var nrow = g.GetTotalRange("SHELL");
```

GetTotals(type*[string]*)

Description

Returns the total number of ’all’, ’list’ and ’range’ rows for a type in a group

Arguments

- **type** (string)

The type of the item

Returns

Array containing number of ’all’, ’list’ and ’range’ rows defined or null if type not in group.

Return type

Array

Example

To get the total number of shell ’all’, ’list’ and ’range’ rows in group g:

```
var totals = g.GetTotals("SHELL");
var nall   = totals[0];
var nlist  = totals[1];
var nrange = totals[2];
```

GetType(row*[integer]*)

Description

Returns the type for an entry in a group

Arguments

- **row** (integer)

The entry in the group types that you want the type for. **Note that entries start at 0, not 1**

Returns

The type of the item (string)

Return type

String

Example

To list the types that are present in group g:

```
for (var t=0; t<g.numtypes; t++)
{
    var type = g.GetType(t);
    Message(type);
}
```

Keyword()

Description

Returns the keyword for this group. **Note that a carriage return is not added**. See also Group.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for goup g:

```
var key = g.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the Group. **Note that a carriage return is not added**. See also Group.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for Group g:

```
var cards = g.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last group in the model.

Arguments

- **Model** (Model)

Model to get last group in

Returns

Group object (or null if there are no groups in the model).

Return type

Group

Example

To get the last group in model m:

```
var g = Group.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free group label in the model. Also see Group.FirstFreeLabel(), Group.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free group label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

Group label.

Return type

Number

Example

To get the last free group label in model m:

```
var label = Group.LastFreeLabel(m);
```

Next()

Description

Returns the next group in the model.

Arguments

No arguments

Returns

Group object (or null if there are no more groups in the model).

Return type

Group

Example

To get the group in model m after group g:

```
var g = g.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) group label in the model. Also see Group.FirstFreeLabel(), Group.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free group label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

Group label.

Return type

Number

Example

To get the next free group label in model m:

```
var label = Group.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a group.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only groups from that model can be picked. If the argument is a Flag then only groups that are flagged with *limit* can be selected. If omitted, or null, any groups from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

Group object (or null if not picked)

Return type

Group

Example

To pick a group from model m giving the prompt 'Pick group from screen':

```
var g = Group.Pick('Pick group from screen', m);
```

Previous()

Description

Returns the previous group in the model.

Arguments

No arguments

Returns

Group object (or null if there are no more groups in the model).

Return type

Group

Example

To get the group in model m before group g:

```
var g = g.Previous();
```

RemoveDataAll(type*[string]*, index*[integer]*)

Description

Removes ’all’ data for a given row number and type in the group.

Arguments

- **type** (string)

The type of the item

- **index** (integer)

Index of ’all’ row you want to Remove. **Note that indices start at 0, not 1**.
0 <= index < Group.GetTotalAll()

Returns

No return value

Example

To remove the data for the 3rd SHELL ’all’ row in group g:

```
g.RemoveDataAll("SHELL", 2);
```

RemoveDataList(type*[string]*, index*[integer]*)

Description

Removes ’list’ data for a given row number and type in the group.

Arguments

- **type** (string)

The type of the item

- **index** (integer)

Index of ’list’ row you want to Remove. **Note that indices start at 0, not 1**.
0 <= index < Group.GetTotalList()

Returns

No return value

Example

To remove the data for the 3rd SHELL ’list’ row in group g:

```
g.RemoveDataList("SHELL", 2);
```

RemoveDataRange(type*[string]*, index*[integer]*)

Description

Removes 'range' data for a given row number and type in the group.

Arguments

- **type** (string)

The type of the item

- **index** (integer)

Index of 'range' row you want to Remove. **Note that indices start at 0, not 1**.
0 <= index < Group.GetTotalRange()

Returns

No return value

Example

To remove the data for the 3rd SHELL 'range' row in group g:

```
g.RemoveDataRange("SHELL", 2);
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the groups in the model.

Arguments

- **Model** (Model)

Model that all groups will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the groups in model m, from 1000000:

```
Group.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged groups in the model.

Arguments

- **Model** (Model)

Model that all the flagged groups will be renumbered in

- **flag** (Flag)

Flag set on the groups that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the groups in model m flagged with f, from 1000000:

```
Group.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select groups using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting groups

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only groups from that model can be selected. If the argument is a Flag then only groups that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any groups can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of groups selected or null if menu cancelled

Return type

Number

Example

To select groups from model m, flagging those selected with flag f, giving the prompt 'Select groups':

```
Group.Select(f, 'Select groups', m);
```

To select groups, flagging those selected with flag f but limiting selection to groups flagged with flag l, giving the prompt 'Select groups':

```
Group.Select(f, 'Select groups', l);
```

SetDataAll(type*[string]*, index*[integer]*, data*[Array of data]*)

Description

Sets 'all' data for a given row number and type in the group.

Arguments

- **type** (string)

The type of the item

- **index** (integer)

Index of ’all’ row you want the data for. **Note that indices start at 0, not 1**.
0 <= index <= Group.GetTotalAll()

- **data** (Array of data)

An array containing data [Group.ADD or Group.REMOVE, BOX (if defined)].

Returns

No return value

Example

To set the data for the 3rd SHELL ’all’ row in group g to ’add box 10’:

```
var data = [Group.ADD, 10];
g.SetDataAll("SHELL", 2, data);
```

SetDataList(type*[string]*, index*[integer]*, data*[Array of data]*)

Description

Sets ’list’ data for a given row number and type in the group.

Arguments

- **type** (string)

The type of the item

- **index** (integer)

Index of ’list’ row you want the data for. **Note that indices start at 0, not 1**.
0 <= index <= Group.GetTotalList()

- **data** (Array of data)

An array containing data [Group.ADD or Group.REMOVE, ITEM1 (if defined), ITEM2 (if defined), ITEM3 (if defined), ITEM4 (if defined), ITEM5 (if defined), BOX (if defined)].

Returns

No return value

Example

To set the data for the 3rd SHELL ’list’ row in group g to ’add 1 2 box 10’:

```
var data = [Group.ADD, 1, 2, 0, 0, 0, 10];
g.SetDataList("SHELL", 2, data);
```

SetDataRange(type*[string]*, index*[integer]*, data*[Array of data]*)

Description

Sets ’range’ data for a given row number and type in the group.

Arguments

- **type** (string)

The type of the item

- **index** (integer)

Index of ’all’ row you want the data for. **Note that indices start at 0, not 1**.
0 <= index <= Group.GetTotalRange()

- **data** (Array of data)

An array containing data [Group.ADD or Group.REMOVE, START, END, BOX (if defined)].

Returns

No return value

Example

To set the data for the 3rd SHELL 'range' row in group g to 'add 100 200 box 10':

```
var data = [Group.ADD, 100, 200, 10];
g.SetDataRange("SHELL", 2, data);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the group.

Arguments

- **flag** (Flag)

Flag to set on the group

Returns

No return value

Example

To set flag f for group g:

```
g.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the group. The group will be sketched until you either call Group.Unsketch(), Group.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the group is sketched. If omitted redraw is true. If you want to sketch several groups and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch group g:

```
g.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged groups in the model. The groups will be sketched until you either call Group.Unsketch(), Group.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged groups will be sketched in

- **flag** (Flag)

Flag set on the groups that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the groups are sketched. If omitted redraw is true. If you want to sketch flagged groups several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all groups flagged with flag in model m:

```
Group.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of groups in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing groups should be counted. If false or omitted referenced but undefined groups will also be included in the total.

Returns

number of groups

Return type

Number

Example

To get the total number of groups in model m:

```
var total = Group.Total(m);
```

Unblank()

Description

Unblanks the group

Arguments

No arguments

Returns

No return value

Example

To unblank group g:

```
g.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the groups in the model.

Arguments

- **Model** (Model)

Model that all groups will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the groups in model m:

```
Group.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged groups in the model.

Arguments

- **Model** (Model)

Model that the flagged groups will be unblanked in

- **flag** (Flag)

Flag set on the groups that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the groups in model m flagged with f:

```
Group.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the groups in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all groups will be unset in

- **flag** (Flag)

Flag to unset on the groups

Returns

No return value

Example

To unset the flag f on all the groups in model m:

```
Group.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the group.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the group is unsketched. If omitted redraw is true. If you want to unsketch several groups and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch group g:

```
g.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all groups.

Arguments

- **Model** (Model)

Model that all groups will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the groups are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all groups in model m:

```
Group.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged groups in the model.

Arguments

- **Model** (Model)

Model that all groups will be unsketched in

- **flag** (Flag)

Flag set on the groups that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the groups are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all groups flagged with flag in model m:

```
Group.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

Group object.

Return type

Group

Example

To check if Group property g.example is a parameter by using the Group.GetParameter() method:

```
if (g.ViewParameters().GetParameter(g.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for group with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The group object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for group g:

```
g.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for group g:

```
g.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for group. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for group g:

```
g.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this group.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for group g:

```
var xrefs = g.Xrefs();
```

---

toString()

Description

Creates a string containing the Group data in keyword format. Note that this contains the keyword header and the keyword cards. See also Group.Keyword() and Group.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for Group g in keyword format

```
var s = g.toString();
```

Image class

The Image class enables writing bitmaps to file. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- WriteBMP(filename*[string]*, resolution (optional)*[constant]*, write_8bit (optional)*[boolean]*, options (optional)*[constant]*)
- WriteGIF(filename*[string]*, resolution (optional)*[constant]*, palette (optional)*[constant]*)
- WriteJPEG(filename*[string]*, resolution (optional)*[constant]*, quality (optional)*[integer]*)
- WritePNG(filename*[string]*, resolution (optional)*[constant]*, write_8bit (optional)*[boolean]*, palette (optional)*[constant]*)

Image constants

Name	Description
Image.COMPRESS	If compression is done for 8 bit bmp images.
Image.DITHER	If dithering is done for 8 bit images.
Image.OPTIMISE	If palette optimisation is done for 8 bit images.
Image.SCREEN	Image will be created at screen resolution.
Image.X2	Image will be created at 2x screen resolution.
Image.X4	Image will be created at 4x screen resolution.

Detailed Description

The Image class enables you to write BMP, GIF, JPEG or PNG images from PRIMER. See the documentation below for more details.

Details of functions

WriteBMP(filename*[string]*, resolution (optional)*[constant]*, write_8bit (optional)*[boolean]*, options (optional)*[constant]*) [static]

Description

Create a bmp image of the current screen image

Arguments

- **filename** (string)

Filename you want to write. The file will be overwritten if it already exists.

- **resolution (optional)** (constant)

The resolution to write the image at. Can be Image.SCREEN, Image.X2 or Image.X4. If omitted screen resolution will be used

- **write_8bit (optional)** (boolean)

BMP images can be written using either 8 bit (256 colours) or 24 bit (16 million colours). If this is true then an 8 bit image will be written. If false (or omitted) a 24 bit image will be written.

- **options (optional)** (constant)

For 8 bit images (see 'write_8bit' argument) the palette can be optimised (Image.OPTIMISE) and/or dithered (Image.DITHER) and/or compressed (Image.COMPRESS) If 0 (or omitted) no palette optimising, dithering or compression will be done.

Returns

No return value

Example

To create a 24 bit png file "/data/test/image.png" at 2x screen resolution

```
Image.WriteBMP("/data/test/image.bmp", Image.X2);
```

WriteGIF(filename*[string]*, resolution (optional)*[constant]*, palette (optional)*[constant]*) [static]

Description

Create a gif image of the current screen image

Arguments

- **filename** (string)

Filename you want to write. The file will be overwritten if it already exists.

- **resolution (optional)** (constant)

The resolution to write the image at. Can be Image.SCREEN, Image.X2 or Image.X4. If omitted screen resolution will be used

- **palette (optional)** (constant)

The palette can be optimised (Image.OPTIMISE) and/or dithered (Image.DITHER). If 0 (or omitted) no palette optimising or dithering will be done.

Returns

No return value

Example

To create a gif file "/data/test/image.gif" at 2x screen resolution

```
Image.WriteGIF("/data/test/image.gif", Image.X2);
```

WriteJPEG(filename*[string]*, resolution (optional)*[constant]*, quality (optional)*[integer]*) [static]

Description

Create a jpeg image of the current screen image

Arguments

- **filename** (string)

Filename you want to write. The file will be overwritten if it already exists.

- **resolution (optional)** (constant)

The resolution to write the image at. Can be Image.SCREEN, Image.X2 or Image.X4. If omitted screen resolution will be used

- **quality (optional)** (integer)

Quality of the image in percent. Can be in the range [10,100]. If omitted, the quality is 90.

Returns

No return value

Example

To create a jpeg file "/data/test/image.jpg" at 2x screen resolution

```
Image.WriteJPEG("/data/test/image.jpg", Image.X2);
```

WritePNG(filename*[string]*, resolution (optional)*[constant]*, write_8bit (optional)*[boolean]*, palette (optional)*[constant]*) [static]

Description

Create a png image of the current screen image

Arguments

- **filename** (string)

Filename you want to write. The file will be overwritten if it already exists.

- **resolution (optional)** (constant)

The resolution to write the image at. Can be Image.SCREEN, Image.X2 or Image.X4. If omitted screen resolution will be used

- **write_8bit (optional)** (boolean)

PNG images can be written using either 8 bit (256 colours) or 24 bit (16 million colours). If this is true then an 8 bit image will be written. If false (or omitted) a 24 bit image will be written.

- **palette (optional)** (constant)

For 8 bit images (see 'write_8bit' argument) the palette can be optimised (Image.OPTIMISE) and/or dithered (Image.DITHER). If 0 (or omitted) no palette optimising or dithering will be done.

Returns

No return value

Example

To create a 24 bit png file "/data/test/image.png" at 2x screen resolution

```
Image.WritePNG("/data/test/image.png", Image.X2);
```

Link class

The Link class allows you to link to other Oasys LS-DYNA Environment programs on the computer. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- ConnectPending()

Member functions

- Send(data*[ArrayBuffer or string]*)
- Terminate()

Link constants

Constants for programs

Name	Description
Link.D3PLOT	Link to D3PLOT
Link.PRIMER	Link to PRIMER
Link.THIS	Link to T-HIS

Constants for status

Name	Description
Link.CONNECTED	Link is connected to the remote script
Link.CREATED	Link has been created but not connected to the remote script
Link.DISCONNECTED	Link has been disconnected as the remote script has finished/exited
Link.TERMINATED	Link has been terminated as the remote program has closed/crashed

Link properties

Name	Type	Description
onDisconnect	function	Function to call when when the link is disconnected, either because the remote program has terminated, or the remote script has finished. The status property can be used to distinguish these cases. The function has no arguments. The Link object is accessible in the function using the 'this' keyword. To unset the function, set the property to null.

onReceipt	function	Function to call to acknowledge receipt of the data in the remote program (data sent from a Send command). The function takes a single argument which is the UUID returned from the original Send command. Note that this function will only be called if the link is asynchronous. In the synchronous case the local program will wait for the remote program to process the Send command so this is unnecessary. To unset the function, set the property to null.
onReceive	function	Function to call when data is received from a Send command in the remote program. The function takes a single argument which contains the data passed by Send. To unset the function, set the property to null.
size (read only)	integer	Size of shared memory buffer in KB
status (read only)	constant	The status of the connection. Can be Link.CREATED, Link.CONNECTED, Link.DISCONNECTED or Link.TERMINATED
synchronous	boolean	whether link Send commands are synchronous or asynchronous (default)
timeout	integer	Timeout for Send commands in msecs. If equal to 0 (default) then a default timeout of 5s is used

Detailed Description

The Link class gives you simple functions for a script to pass information to and from scripts running on the other Oasys LS-DYNA Environment programs on the same computer.
It is the JavaScript equivalent of the user pressing the 'Post' button in PRIMER to start D3PLOT or T/HIS, or the user pressing 'PRIMER' from D3PLOT or T/HIS to start PRIMER, and then passing data between the programs. If the user has not already started the remote program by doing this manually, the Link class will start the remote program.
The following simple example shows a script that is running in PRIMER, starting a link to D3PLOT. Once the link is established, the scripts running in PRIMER and D3PLOT can pass string data to the other program. This data is then displayed in a window on the remote program.
Note that just like when starting the link from PRIMER to D3PLOT manually, to start the link from the script, a model must already be loaded into PRIMER that also has results files for D3PLOT.
Script 'primer.js' running in PRIMER:

```
// Global scope for link variable
var link;
// Get path for this script, and hence path for d3plot.js
var scriptD3PLOT = arguments[0].replace('primer.js', 'd3plot.js');
// Create a simple GUI
Window.Theme(Window.THEME_CURRENT);
var w = new Window("Link PRIMER", 0.9, 1.0, 0.9, 1.0);
var start = new Widget(w, Widget.BUTTON, 1, 40, 1, 7, "Start D3PLOT");
start.onClick = startD3PLOT;
var msg  = new Widget(w, Widget.TEXTBOX,  1, 50, 8, 14, "");
var send = new Widget(w, Widget.BUTTON,  51, 90, 8, 14, "Send Message");
send.onClick = sendMessage;
send.active = false;
w.onClose = Exit;
w.Show(false);
////////////////////////////////////////////////////////////////////////////
function startD3PLOT()
{
// Function to start D3PLOT and create link
// object to pass to Link constructor
    var obj = {
               script:             scriptD3PLOT,
               connectFunction:    "connectD3PLOT",
               receiveFunction:    "receiveD3PLOT",
               disconnectFunction: "Exit",
               onReceive:          receivePRIMER,
               onDisconnect:       Exit,
              };
// Create link and save to global scope for this simple example
    link = new Link(Link.D3PLOT, obj);
// Do NOT try to immediately send something to D3PLOT using the link
// It may not be properly initialised yet.
// See functions connectD3PLOT and receivePRIMER
// Make the 'Send Message' button active and disable the 'Start D3PLOT' button
```

```
    start.active = false;
    send.active  = true;
}
////////////////////////////////////////////////////////////////////////////
function receivePRIMER(data)
{
// Function called when data is received from D3PLOT across the link
// If the object has a property called 'connect' then D3PLOT has sent
// this from the connect function and so is ready to receive data.
    if (data.connect)
    {
        Message("D3PLOT is connected");
// We could now send information to D3PLOT if needed
//      link.Send(...);
    }
// If the object has a property called 'string' then it is the string
// from the message box
    else if (data.string)
    {
        Window.Message("Example string from D3PLOT", data.string );
    }
}
////////////////////////////////////////////////////////////////////////////
function sendMessage()
{
// Sends whatever is in textbox when 'Send Message' is pressed, to D3PLOT
    var uuid = link.Send( { string: msg.text } );
}
```

Script 'd3plot.js' running in D3PLOT:

```
// Global scope for link variable
var link;
// Create a simple GUI
Window.Theme(Window.THEME_CURRENT);
var w = new Window("Link D3PLOT", 0.9, 1.0, 0.9, 1.0);
var msg  = new Widget(w, Widget.TEXTBOX,  1, 50, 1, 7, "");
var send = new Widget(w, Widget.BUTTON,  51, 90, 1, 7, "Send Message");
send.onClick = sendMessage;
send.active  = false;
w.onClose = Exit;
w.Show(false);
////////////////////////////////////////////////////////////////////////////
function connectD3PLOT(l)
{
// Function called when the connection from PRIMER is made
// Save link to global scope for this simple example
    link = l;
// Make the 'Send Message' button active
    send.active = true;
// Send a command back to PRIMER to say that we are connected
    link.Send( { connect: "CONNECTED" } );
}
////////////////////////////////////////////////////////////////////////////
function receiveD3PLOT(data)
{
// Function called when data is received from PRIMER across the link
// If the object has a property called 'string' then it is the string
// from the message box
    if (data.string)
    {
        Window.Message("Example string from PRIMER", data.string);
    }
}
////////////////////////////////////////////////////////////////////////////
function sendMessage()
{
// Sends whatever is in textbox when 'Send Message' is pressed, to PRIMER
    var uuid = link.Send( { string: msg.text } );
}
```

Constructor

new Link(program*[constant]*, data*[object]*)

Description

Create a new Link object to communicate with one of the linked Oasys LS-DYNA programs running on the computer, starting it if required.
Note that you should **not** try to use the Send method immediately after the constructor. If you do, the remote program may not have time to initialise properly. If you want to interact with the remote program immediately then the connect function in the remote program should send a command to the local program to say that it is initialised. The onReceive function in the local program can then be used to send the command. See the simple example at the top of this class for more details.
For the specific case of D3PLOT starting T/HIS, the connect function will not be called straight away. As D3PLOT and T/HIS are integrated together into a single user interface, the script **must** return back to the main event loop after the constructor is called so that D3PLOT and T/HIS can communicate with each other to set up the integration. Once this has been done and D3PLOT is 'idle', the connect function will be called. The idleTime property in the data object can be used to control this delay if required. As D3PLOT must return back to the event loop when starting T/HIS do not put any statements immediately after the link constructor call.

Arguments

- **program** (constant)

The program that you want to link to. See the programs constants

- **data** (object)

link data

Object has the following properties:

Name	Type	Description
connectFunction	string	Name of function to call in the remote script when the link connected. This function will be called when the connection is made with a Link object as the argument.
disconnectFunction (optional)	string	Name of function to call in the remote script when the link is disconnected, either because the local program has terminated, or the local script has finished. The status property can be used to distinguish these cases. This is provided as an easy way to set up a disconnect callback in the remote script. Alternatively, the remote script could set the onDisconnect property in connectFunction.
idelTime (optional)	integer	For the specific case of starting T/HIS from D3PLOT, the required time for D3PLOT to be idle for in msecs before the connect function is called. If omitted then a default idle time of 1s is used
model (optional)	integer	Model number to pass across the link to the remote program
onDisconnect (optional)	function	Function to call in the local script when the link is disconnected, either because the remote program has terminated, or the remote script has finished. The status property can be used to distinguish these cases.
onReceipt (optional)	function	Function to call in the local script to acknowledge receipt of the data in the remote program (data sent from a Send command in the local program to the remote program). The function takes a single argument which is the UUID returned from the original Send command. This function is useful if the link is asynchronous to know that the command has been processed. In the synchronous case the local program will wait for the remote program to process the Send command so this is unnecessary. However, the function can still be used.
onReceive (optional)	function	Function to call in the local script when data is received from a Send command in the remote script. The function takes a single argument which contains the data passed by Send
path (optional)	string	Path to a model to pass across the link to the remote program. This can be used instead of a model. For example if you want to open a different model in D3PLOT to what you have open in PRIMER then you could pass the path to the d3plot file to open

receiptFunction (optional)	string	Name of function to call in the remote script to acknowledge receipt of the data in the local script (data sent from a Send command in the remote link object to the local script). The function takes a single argument which is the UUID returned from the original Send command. Note that this function will only be called if the link is asynchronous. In the synchronous case the local program will wait for the remote to process the Send command so this is unnecessary. This is provided as an easy way to set up a receipt callback in the remote script. Alternatively, the remote script could set the onReceipt property in connectFunction.
receiveFunction (optional)	string	Name of function to call in the remote script when data is received from a Send command in the local script. The function takes a single argument which contains the data passed by Send. This is provided as an easy way to set up a receive callback in the remote script. Alternatively, the remote script could set the onReceive property in connectFunction.
script	string	The script to run
size (optional)	integer	Size of shared memory buffer in KB. If omitted a default size of 10KB is used
synchronous (optional)	boolean	Whether link Send commands are synchronous or asynchronous (default if omitted)
timeout (optional)	integer	Timeout for Send commands in msecs. If equal to 0 (default if omitted) then a default timeout of 5s is used

Returns

Link object

Return type

Link

Example

To create a connection to D3PLOT, calling function 'myConnectFunc' in remote script '/example/script.js' when the connection is made:

```
var s = new Link(Link.D3PLOT, { script: '/example/script.js', connectFunction: 'myConnectFunc' } );
```

Details of functions

ConnectPending() [static]

Description

Check if the script has been started on the remote side of a link and, if so, whether calling the connect function is pending. This may be useful to make a script behave differently, depending on whether it is being run normally or has been started on the remote side of a link.

Arguments

No arguments

Returns

true if a remote link script is waiting to connect, false for all other cases

Return type

boolean

Example

To check if the script has been started on the remote side of a link:

```
if (Link.ConnectPending() ) { do something ... }
```

Send(data*[ArrayBuffer or string]*)

Description

Send data/command across link

Arguments

- **data** (ArrayBuffer or string)

The data/command to pass. The data can either be an ArrayBuffer or any JavaScript type that can be represented as JSON

Returns

UUID string to uniquely identify this Send command

Return type

String

Example

To send some data across link l:

```
var data = { action: "read", filename: "example.ptf" };
l.Send(data);
```

Terminate()

Description

Terminate the program on the remote side of the link

Arguments

No arguments

Returns

No return value

Example

To terminate the program on the remote side of link l:

```
l.Terminate();
```

Mechanism class

The Mechanism class gives you access to mechanism cards in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AddNodeSetToAssembly(index*[integer]*, nsid*[integer]*)
- AddPartSetToAssembly(index*[integer]*, psid*[integer]*)
- AddPartToAssembly(index*[integer]*, pid*[integer]*)
- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetAssembly(index*[integer]*)
- GetComments()
- GetConnection(index*[integer]*)
- GetParameter(prop*[string]*)
- GetPoint(index*[integer]*)
- GetPointData(rpt*[integer]*)
- GetPointTitle(rpt*[integer]*)
- Next()
- Previous()
- RemoveConnection(index*[integer]*)
- RemoveNodeSetFromAssembly(index*[integer]*, nsid*[integer]*)
- RemovePartFromAssembly(index*[integer]*, pid*[integer]*)
- RemovePartSetFromAssembly(index*[integer]*, psid*[integer]*)

- RemovePoint(index*[integer]*)
- SetConnection(index*[integer]*, data*[object]*)
- SetFlag(flag*[Flag]*)
- SetPoint(index*[integer]*, data*[object]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()

Mechanism constants

Constants for Connection types

Name	Description
Mechanism.COUPLER	Coupler mechanism connection
Mechanism.HINGE	Hinge mechanism connection
Mechanism.LINE	Line mechanism connection
Mechanism.PIN	Pin mechanism connection

Constants for Coupler modes

Name	Description
Mechanism.ROTATION	Rotational coupling on mechanism coupler
Mechanism.TRANSLATION	Translational coupling on mechanism coupler

Mechanism properties

Name	Type	Description
assemblies (read only)	integer	Number of assemblies defined.
connections (read only)	integer	Number of connections defined.
exists (read only)	logical	true if mechanism exists, false if referred to but not defined.
id (read only)	integer	Mechanism number. Also see the label property which is an alternative name for this.
include	integer	The Include file number that the mechanism is in.
label (read only)	integer	Mechanism number. Also see the id property which is an alternative name for this.
model (read only)	integer	The Model number that the mechanism is in.
points (read only)	integer	Number of reference points defined.
title	string	Mechanism title.

Detailed Description

The Mechanism class allows you to create, modify, edit and manipulate mechanism cards. See the documentation below for more details.

Details of functions

AddNodeSetToAssembly(index*[integer]*, nsid*[integer]*)

Description

Add node set to assembly

Arguments

- **index** (integer)

The index of the assembly in which you want to add node set. **Note that reference points start at 0, not 1**. 0 <= index < assemblies

- **nsid** (integer)

The node set ID that you want to add.

Returns

No return value

Example

To add node set 3 in 3rd assembly in mechanism m:

```
m.AddNodeSetToAssembly(2,3);
```

AddPartSetToAssembly(index*[integer]*, psid*[integer]*)

Description

Add part set to assembly

Arguments

- **index** (integer)

The index of the assembly in which you want to add part set. **Note that reference points start at 0, not 1**. 0 <= index < assemblies

- **psid** (integer)

The part set ID that you want to add.

Returns

No return value

Example

To add part set 3 in 3rd assembly in mechanism m:

```
m.AddPartSetToAssembly(2,3);
```

AddPartToAssembly(index*[integer]*, pid*[integer]*)

Description

Add part to assembly

Arguments

- **index** (integer)

The index of the assembly in which you want to add part. **Note that reference points start at 0, not 1**. 0 <= index < assemblies

- **pid** (integer)

The part ID that you want to add.

Returns

No return value

Example

To add part 3 in 3rd assembly in mechanism m:

```
m.AddPartToAssembly(2,3);
```

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a mechanism.

Arguments

- **Comment** (Comment)

Comment that will be attached to the mechanism

Returns

No return value

Example

To associate comment c to the mechanism m:

```
m.AssociateComment(c);
```

Blank()

Description

Blanks the mechanism

Arguments

No arguments

Returns

No return value

Example

To blank mechanism m:

```
m.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the mechanisms in the model.

Arguments

- **Model** (Model)

Model that all mechanisms will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the mechanisms in model m:

```
Mechanism.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged mechanisms in the model.

Arguments

- **Model** (Model)

Model that all the flagged mechanisms will be blanked in

- **flag** (Flag)

Flag set on the mechanisms that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the mechanisms in model m flagged with f:

```
Mechanism.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the mechanism is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if mechanism m is blanked:

```
if (m.Blanked() ) do_something...
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the mechanism.

Arguments

- **flag** (Flag)

Flag to clear on the mechanism

Returns

No return value

Example

To clear flag f for mechanism m:

```
m.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the mechanism. The target include of the copied mechanism can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

Mechanism object

Return type

Mechanism

Example

To copy mechanism m into mechanism z:

```
var z = m.Copy();
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a mechanism.

Arguments

- **Comment** (Comment)

Comment that will be detached from the mechanism

Returns

No return value

Example

To detach comment c from the mechanism m:

```
m.DetachComment(c);
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for mechanism with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The mechanism object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for mechanism m:

```
m.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for mechanism m:

```
m.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for mechanism. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for mechanism m:

```
m.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first mechanism in the model.

Arguments

- **Model** (Model)

Model to get first mechanism in

Returns

Mechanism object (or null if there are no mechanisms in the model).

Return type

Mechanism

Example

To get the first mechanism in model m:

```
var m = Mechanism.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free mechanism label in the model. Also see Mechanism.LastFreeLabel(), Mechanism.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free mechanism label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

Mechanism label.

Return type

Number

Example

To get the first free mechanism label in model m:

```
var label = Mechanism.FirstFreeLabel(m);
```

FlagAll(Model[*Model*], flag[*Flag*]) [static]

Description

Flags all of the mechanisms in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all mechanisms will be flagged in

- **flag** (Flag)

Flag to set on the mechanisms

Returns

No return value

Example

To flag all of the mechanisms with flag f in model m:

```
Mechanism.FlagAll(m, f);
```

Flagged(flag[*Flag*])

Description

Checks if the mechanism is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the mechanism

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if mechanism m has flag f set on it:

```
if (m.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each mechanism in the model.
Note that ForEach has been designed to make looping over mechanisms as fast as possible and so has some limitations.
Firstly, a single temporary Mechanism object is created and on each function call it is updated with the current mechanism data. This means that you should not try to store the Mechanism object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new mechanisms inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all mechanisms are in

- **func** (function)

Function to call for each mechanism

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the mechanisms in model m:

```
Mechanism.ForEach(m, test);
function test(m)
{
// m is Mechanism object
}
```

To call function test for all of the mechanisms in model m with optional object:

```
var data = { x:0, y:0 };
Mechanism.ForEach(m, test, data);
function test(m, extra)
{
// m is Mechanism object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Mechanism objects or properties for all of the mechanisms in a model in PRIMER. If the optional property argument is not given then an array of Mechanism objects is returned. If the property argument is given, that property value for each mechanism is returned in the array instead of a Mechanism object

Arguments

- **Model** (Model)

Model to get mechanisms from

- **property (optional)** (string)

Name for property to get for all mechanisms in the model

Returns

Array of Mechanism objects or properties

Return type

Array

Example

To make an array of Mechanism objects for all of the mechanisms in model m:

```
var a = Mechanism.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each mechanism in model m:

```
var a = Mechanism.GetAll(m, 'foo');
```

GetAssembly(index*[integer]*)

Description

Returns the information for an assembly

Arguments

- **index** (integer)

The index of the assembly you want the coordinates for. **Note that reference points start at 0, not 1**. 0 <= index < assemblies

Returns

Object with the following properties:

Name	Type	Description
label	integer	Assembly label
parent	integer	Parent assembly label
title	string	Assembly title

Return type

object

Example

To get the information for the 3rd assembly for mechanism m:

```
var info = m.GetAssembly(2);
```

GetComments()

Description

Extracts the comments associated to a mechanism.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the mechanism m:

```
var comm_array = m.GetComments();
```

GetConnection(index*[integer]*)

Description

Returns the information for a connection

Arguments

- **index** (integer)

The index of the connection you want the information for. **Note that connections start at 0, not 1**. 0 <= index < connections

Returns

Object with the following properties:

Name	Type	Description
angle	real	Current angle in degrees (for Mechanism.LINE and Mechanism.HINGE)
assembly1	integer	Assembly 1 label
assembly2	integer	Assembly 2 label
assembly3	integer	Assembly 3 label
coefficient1	real	Coefficient for linear coupler equation for connection 1 (for Mechanism.COUPLER)
coefficient2	real	Coefficient for linear coupler equation for connection 2 (for Mechanism.COUPLER)
coefficient3	real	Coefficient for linear coupler equation for connection 3 (for Mechanism.COUPLER)
connection1	integer	Connection 1 label (for Mechanism.COUPLER)
connection2	integer	Connection 2 label (for Mechanism.COUPLER)
connection3	integer	Connection 3 label (for Mechanism.COUPLER)
distance	real	Current distance (for Mechanism.LINE)
factor1	real	Factor 1 on Assembly 3 (Mechanism.LINE only)
factor2	real	Factor 2 on Assembly 3 (Mechanism.LINE only)
label	integer	Connection label
locked	integer	1 if locked (for Mechanism.LINE, Mechanism.PIN and Mechanism.HINGE)
mode1	integer	Coupling mode for connection 1. 0 = translational coupling, 1 = rotational coupling (for Mechanism.COUPLER)
mode2	integer	Coupling mode for connection 2. 0 = translational coupling, 1 = rotational coupling (for Mechanism.COUPLER)
mode3	integer	Coupling mode for connection 3. 0 = translational coupling, 1 = rotational coupling (for Mechanism.COUPLER)

node1	integer	Node 1 label (for Mechanism.LINE, Mechanism.PIN and Mechanism.HINGE)
node2	integer	Node 2 label (for Mechanism.LINE, Mechanism.PIN and Mechanism.HINGE)
nrotation	real	-ve rotation limit in degrees (for Mechanism.LINE and Mechanism.HINGE)
nslide	real	-ve slide translation (for Mechanism.LINE)
protation	real	+ve rotation limit in degrees (for Mechanism.LINE and Mechanism.HINGE)
pslide	real	+ve slide translation (for Mechanism.LINE)
title	string	Connection label
type	integer	Mechanism type (Mechanism.COUPLER, Mechanism.HINGE, Mechanism.LINE, Mechanism.PIN)
x1	real	X1 coordinates (for Mechanism.LINE, Mechanism.PIN and Mechanism.HINGE)
x2	real	X2 coordinates (for Mechanism.LINE, Mechanism.PIN and Mechanism.HINGE)
y1	real	Y1 coordinates (for Mechanism.LINE, Mechanism.PIN and Mechanism.HINGE)
y2	real	Y2 coordinates (for Mechanism.LINE, Mechanism.PIN and Mechanism.HINGE)
z1	real	Z1 coordinates (for Mechanism.LINE, Mechanism.PIN and Mechanism.HINGE)
z2	real	Z2 coordinates (for Mechanism.LINE, Mechanism.PIN and Mechanism.HINGE)

Return type

object

Example

To get the information for the 3rd connection for mechanism m:

```
var info = m.GetConnection(2);
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of Mechanism objects for all of the flagged mechanisms in a model in PRIMER If the optional property argument is not given then an array of Mechanism objects is returned. If the property argument is given, then that property value for each mechanism is returned in the array instead of a Mechanism object

Arguments

- **Model** (Model)

Model to get mechanisms from

- **flag** (Flag)

Flag set on the mechanisms that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged mechanisms in the model

Returns

Array of Mechanism objects or properties

Return type

Array

Example

To make an array of Mechanism objects for all of the mechanisms in model m flagged with f:

```
var m = Mechanism.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the mechanisms in model m flagged with f:

```
var a = Mechanism.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the Mechanism object for a mechanism ID.

Arguments

- **Model** (Model)

Model to find the mechanism in

- **number** (integer)

number of the mechanism you want the Mechanism object for

Returns

Mechanism object (or null if mechanism does not exist).

Return type

Mechanism

Example

To get the Mechanism object for mechanism 100 in model m

```
var m = Mechanism.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a Mechanism property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the Mechanism.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

mechanism property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if Mechanism property m.example is a parameter:

```
Options.property_parameter_names = true;
if (m.GetParameter(m.example) ) do_something...
Options.property_parameter_names = false;
```

To check if Mechanism property m.example is a parameter by using the GetParameter method:

```
if (m.ViewParameters().GetParameter(m.example) ) do_something...
```

GetPoint(index*[integer]*)

Description

Returns the information for a reference point

Arguments

- **index** (integer)

The index of the reference point you want the information for. **Note that reference points start at 0, not 1**. 0 <= index < points

Returns

Object with the following properties:

Name	Type	Description
assembly	integer	Assembly label
csys	integer	Coordinate system
hpt	boolean	If point has been automatically created by PRIMER at the H-point
label	integer	Point label
node	integer	Node label (0 if coordinate)
rx	boolean	Point restrained rotationally in X
ry	boolean	Point restrained rotationally in Y
rz	boolean	Point restrained rotationally in Z
title	string	Point title
tx	boolean	Point restrained translationally in X
ty	boolean	Point restrained translationally in Y
tz	boolean	Point restrained translationally in Z
x	real	Node/point x coordinate
y	real	Node/point y coordinate
z	real	Node/point z coordinate

Return type

object

Example

To get the information for the 3rd reference point for mechanism m:

```
var info = m.GetPoint(2);
```

GetPointData(rpt*[integer]*)

Description

Returns the coordinates of a reference point

Arguments

- **rpt** (integer)

The reference point you want the coordinates for. **Note that reference points start at 0, not 1**.

Returns

Array containing the reference point coordinates

Return type

Array

Example

To get the coordinates of the 3rd reference point for mechanism mec:

```
var c = mec.GetPointData(2)
```

GetPointTitle(rpt*[integer]*)

Description

Returns the title of a reference point

Arguments

- **rpt** (integer)

The reference point you want the title for. **Note that reference points start at 0, not 1**.

Returns

The reference point title

Return type

String

Example

To get the title of the 3rd reference point for mechanism mec:

```
var c = mec.GetPointTitle(2)
```

Last(Model*[Model]*) [static]

Description

Returns the last mechanism in the model.

Arguments

- **Model** (Model)

Model to get last mechanism in

Returns

Mechanism object (or null if there are no mechanisms in the model).

Return type

Mechanism

Example

To get the last mechanism in model m:

```
var m = Mechanism.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free mechanism label in the model. Also see Mechanism.FirstFreeLabel(), Mechanism.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free mechanism label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

Mechanism label.

Return type

Number

Example

To get the last free mechanism label in model m:

```
var label = Mechanism.LastFreeLabel(m);
```

Next()

Description

Returns the next mechanism in the model.

Arguments

No arguments

Returns

Mechanism object (or null if there are no more mechanisms in the model).

Return type

Mechanism

Example

To get the mechanism in model m after mechanism m:

```
var m = m.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) mechanism label in the model. Also see Mechanism.FirstFreeLabel(), Mechanism.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free mechanism label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

Mechanism label.

Return type

Number

Example

To get the next free mechanism label in model m:

```
var label = Mechanism.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a mechanism.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only mechanisms from that model can be picked. If the argument is a Flag then only mechanisms that are flagged with *limit* can be selected. If omitted, or null, any mechanisms from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

Mechanism object (or null if not picked)

Return type

Mechanism

Example

To pick a mechanism from model m giving the prompt 'Pick mechanism from screen':

```
var m = Mechanism.Pick('Pick mechanism from screen', m);
```

Previous()

Description

Returns the previous mechanism in the model.

Arguments

No arguments

Returns

Mechanism object (or null if there are no more mechanisms in the model).

Return type

Mechanism

Example

To get the mechanism in model m before mechanism m:

```
var m = m.Previous();
```

RemoveConnection(index*[integer]*)

Description

Removes a connection from a mechanism

Arguments

- **index** (integer)

The index of the connection you want to remove. **Note that connections start at 0, not 1**. 0 <= index < connections

Returns

no return value

Example

To remove the 3rd connection for mechanism m:

```
m.RemoveConnection(2);
```

RemoveNodeSetFromAssembly(index*[integer]*, nsid*[integer]*)

Description

Remove node set from assembly

Arguments

- **index** (integer)

The index of the assembly from which you want to remove the node set. **Note that reference points start at 0, not 1**. 0 <= index < assemblies

- **nsid** (integer)

The node set ID that you want to remove.

Returns

No return value

Example

To remove node set 3 from 3rd assembly in mechanism m:

```
m.RemoveNodeSetFromAssembly(2,3);
```

RemovePartFromAssembly(index*[integer]*, pid*[integer]*)

Description

Remove part from assembly

Arguments

- **index** (integer)

The index of the assembly from which you want to remove the part. **Note that reference points start at 0, not 1**. 0 <= index < assemblies

- **pid** (integer)

The part ID that you want to remove.

Returns

No return value

Example

To remove part 3 from 3rd assembly in mechanism m:

```
m.RemovePartFromAssembly(2,3);
```

RemovePartSetFromAssembly(index*[integer]*, psid*[integer]*)

Description

Remove part set from assembly

Arguments

- **index** (integer)

The index of the assembly from which you want to remove the part set. **Note that reference points start at 0, not 1**. 0 <= index < assemblies

- **psid** (integer)

The part set ID that you want to remove.

Returns

No return value

Example

To remove part set 3 from 3rd assembly in mechanism m:

```
m.RemovePartSetFromAssembly(2,3);
```

RemovePoint(index*[integer]*)

Description

Removes a reference point from a mechanism

Arguments

- **index** (integer)

The index of the reference point you want to remove. **Note that reference points start at 0, not 1**. 0 <= index < points

Returns

no return value

Example

To remove the 3rd reference point for mechanism m:

```
m.RemovePoint(2);
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the mechanisms in the model.

Arguments

- **Model** (Model)

Model that all mechanisms will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the mechanisms in model m, from 1000000:

```
Mechanism.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged mechanisms in the model.

Arguments

- **Model** (Model)

Model that all the flagged mechanisms will be renumbered in

- **flag** (Flag)

Flag set on the mechanisms that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the mechanisms in model m flagged with f, from 1000000:

```
Mechanism.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select mechanisms using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting mechanisms

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only mechanisms from that model can be selected. If the argument is a Flag then only mechanisms that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any mechanisms can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of mechanisms selected or null if menu cancelled

Return type

Number

Example

To select mechanisms from model m, flagging those selected with flag f, giving the prompt 'Select mechanisms':

```
Mechanism.Select(f, 'Select mechanisms', m);
```

To select mechanisms, flagging those selected with flag f but limiting selection to mechanisms flagged with flag l, giving the prompt 'Select mechanisms':

```
Mechanism.Select(f, 'Select mechanisms', l);
```

SetConnection(index*[integer]*, data*[object]*)

Description

Sets the data for a connection in a mechanism

Arguments

- **index** (integer)

The index of the connection you want to set. **Note that connections start at 0, not 1**. To add a new connection use index connections

- **data** (object)

Object containing the connection data. The properties can be:

Object has the following properties:

Name	Type	Description
angle (optional)	real	Current angle in degrees (for Mechanism.LINE and Mechanism.HINGE)
assembly1 (optional)	integer	Assembly 1 label (required for Mechanism.LINE, Mechanism.PIN and Mechanism.HINGE)
assembly2 (optional)	integer	Assembly 2 label (required for Mechanism.LINE, Mechanism.PIN and Mechanism.HINGE)
assembly3	integer	Assembly 3 label (required for Mechanism.LINE)
coefficient1 (optional)	real	Coefficient for linear coupler equation for connection 1 (for Mechanism.COUPLER)
coefficient2 (optional)	real	Coefficient for linear coupler equation for connection 2 (for Mechanism.COUPLER)
coefficient3 (optional)	real	Coefficient for linear coupler equation for connection 3 (for Mechanism.COUPLER)
connection1 (optional)	integer	Connection 1 label (for Mechanism.COUPLER)
connection2 (optional)	integer	Connection 2 label (for Mechanism.COUPLER)
connection3 (optional)	integer	Connection 3 label (for Mechanism.COUPLER)
distance (optional)	real	Current distance (for Mechanism.LINE)
factor1 (optional)	real	Factor 1 on Assembly 3 (Mechanism.LINE only)
factor2 (optional)	real	Factor 2 on Assembly 3 (Mechanism.LINE only)
locked (optional)	integer	1 if locked (for Mechanism.LINE, Mechanism.PIN and Mechanism.HINGE)
mode1 (optional)	integer	Coupling mode for connection 1. 0 = translational coupling, 1 = rotational coupling (for Mechanism.COUPLER)
mode2 (optional)	integer	Coupling mode for connection 2. 0 = translational coupling, 1 = rotational coupling (for Mechanism.COUPLER)
mode3 (optional)	integer	Coupling mode for connection 3. 0 = translational coupling, 1 = rotational coupling (for Mechanism.COUPLER)
node1 (optional)	integer	Node 1 label (for Mechanism.LINE, Mechanism.PIN and Mechanism.HINGE, not required if using x1, y1 and z1)
node2 (optional)	integer	Node 2 label (for Mechanism.LINE, Mechanism.PIN and Mechanism.HINGE, not required if using x2, y2 and z2)
nrotation (optional)	real	-ve rotation limit in degrees (for Mechanism.LINE and Mechanism.HINGE)
nslide (optional)	real	-ve slide translation (for Mechanism.LINE)
protation (optional)	real	+ve rotation limit in degrees (for Mechanism.LINE and Mechanism.HINGE)
pslide (optional)	real	+ve slide translation (for Mechanism.LINE)
title (optional)	string	Title
type	integer	Connection type. Can be one of: Mechanism.PIN, Mechanism.LINE, Mechanism.HINGE or Mechanism.COUPLER]
x1 (optional)	real	x1 coordinate (for Mechanism.LINE, Mechanism.PIN and Mechanism.HINGE, not required if using node1)
x2 (optional)	real	x2 coordinate (for Mechanism.LINE, Mechanism.PIN and Mechanism.HINGE, not required if using node2)

y1 (optional)	real	y1 coordinate (for Mechanism.LINE, Mechanism.PIN and Mechanism.HINGE, not required if using node1)
y2 (optional)	real	y2 coordinate (for Mechanism.LINE, Mechanism.PIN and Mechanism.HINGE, not required if using node2)
z1 (optional)	real	z1 coordinate (for Mechanism.LINE, Mechanism.PIN and Mechanism.HINGE, not required if using node1)
z2 (optional)	real	z2 coordinate (for Mechanism.LINE, Mechanism.PIN and Mechanism.HINGE, not required if using node2)

Returns

no return value

Example

To add a new pin connection to mechanism m between assemblies 5 and 6 at node 1000 with title "Example connection":

```
var data = { type:Mechanism.PIN, assembly1:5, assembly2:6, node1:1000,
title:"Example connection" };
m.SetConnection(m.connections, data);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the mechanism.

Arguments

- **flag** (Flag)

Flag to set on the mechanism

Returns

No return value

Example

To set flag f for mechanism m:

```
m.SetFlag(f);
```

SetPoint(index*[integer]*, data*[object]*)

Description

Sets the data for a reference point in a mechanism

Arguments

- **index** (integer)

The index of the reference point you want to set. **Note that reference points start at 0, not 1**. To add a new point use index points

- **data** (object)

Object containing the reference point data. The properties can be:

Object has the following properties:

Name	Type	Description
assembly	integer	Assembly label

csys (optional)	integer	Coordinate system label
node (optional)	integer	Node label (not required if using x, y and z)
rx (optional)	boolean	Point restrained rotationally in X
ry (optional)	boolean	Point restrained rotationally in Y
rz (optional)	boolean	Point restrained rotationally in Z
title (optional)	string	Point title
tx (optional)	boolean	Point restrained translationally in X
ty (optional)	boolean	Point restrained translationally in Y
tz (optional)	boolean	Point restrained translationally in Z
x (optional)	real	x coordinate (not required if using node)
y (optional)	real	y coordinate (not required if using node)
z (optional)	real	z coordinate (not required if using node)

Returns

no return value

Example

To add a new reference point to mechanism m assembly 5 at node 1000 with title "Example point" restrained in x:

```
var data = { assembly:5, node:1000, title:"Example point", tx:true };
m.SetPoint(m.points, data);
```

To add a new reference point to mechanism m assembly 5 at (10, 20, 30) with title "Example point":

```
var data = { assembly:5, x:10, y:20, z:30, title:"Example point" };
m.SetPoint(m.points, data);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the mechanism. The mechanism will be sketched until you either call Mechanism.Unsketch(), Mechanism.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the mechanism is sketched. If omitted redraw is true. If you want to sketch several mechanisms and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch mechanism m:

```
m.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged mechanisms in the model. The mechanisms will be sketched until you either call Mechanism.Unsketch(), Mechanism.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged mechanisms will be sketched in

- **flag** (Flag)

Flag set on the mechanisms that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the mechanisms are sketched. If omitted redraw is true. If you want to sketch flagged mechanisms several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all mechanisms flagged with flag in model m:

```
Mechanism.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of mechanisms in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing mechanisms should be counted. If false or omitted referenced but undefined mechanisms will also be included in the total.

Returns

number of mechanisms

Return type

Number

Example

To get the total number of mechanisms in model m:

```
var total = Mechanism.Total(m);
```

Unblank()

Description

Unblanks the mechanism

Arguments

No arguments

Returns

No return value

Example

To unblank mechanism m:

```
m.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the mechanisms in the model.

Arguments

- **Model** (Model)

Model that all mechanisms will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the mechanisms in model m:

```
Mechanism.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged mechanisms in the model.

Arguments

- **Model** (Model)

Model that the flagged mechanisms will be unblanked in

- **flag** (Flag)

Flag set on the mechanisms that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the mechanisms in model m flagged with f:

```
Mechanism.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the mechanisms in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all mechanisms will be unset in

- **flag** (Flag)

Flag to unset on the mechanisms

Returns

No return value

Example

To unset the flag f on all the mechanisms in model m:

```
Mechanism.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the mechanism.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the mechanism is unsketched. If omitted redraw is true. If you want to unsketch several mechanisms and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch mechanism m:

```
m.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all mechanisms.

Arguments

- **Model** (Model)

Model that all mechanisms will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the mechanisms are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all mechanisms in model m:

```
Mechanism.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged mechanisms in the model.

Arguments

- **Model** (Model)

Model that all mechanisms will be unsketched in

- **flag** (Flag)

Flag set on the mechanisms that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the mechanisms are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all mechanisms flagged with flag in model m:

```
Mechanism.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

Mechanism object.

Return type

Mechanism

Example

To check if Mechanism property m.example is a parameter by using the Mechanism.GetParameter() method:

```
if (m.ViewParameters().GetParameter(m.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for mechanism with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The mechanism object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for mechanism m:

```
m.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for mechanism m:

```
m.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for mechanism. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for mechanism m:

```
m.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this mechanism.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for mechanism m:

```
var xrefs = m.Xrefs();
```

Model class

The Model class gives you access to models in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll()
- DeleteAll()
- First()
- FirstFreeItemLabel(type*[string]*, layer (optional)*[Include number]*)
- GetAll()
- GetFromID(model number*[integer]*)
- Last()
- LastFreeItemLabel(type*[string]*, layer (optional)*[Include number]*)
- Merge(Primary Model*[Model]*, Secondary Model*[Model]*, Option to fix clashes (optional)*[constant]*, Merge nodes flag (optional)*[boolean]*, dist (required if merge nodes flag used) (optional)*[real]*, label (optional)*[integer]*, position (optional)*[integer]*)
- NextFreeItemLabel(type*[string]*, layer (optional)*[Include number]*)
- Read(filename*[string]*, filetype (optional)*[constant]*, number (optional)*[integer]*)
- Select(prompt*[string]*, modal (optional)*[boolean]*)
- Total()
- UnblankAll()

Member functions

- AreaVolumeFlagged(flag*[Flag]*)
- Attached(flag*[Flag]*, redraw (optional)*[boolean]*)
- Autofix()
- Blank()
- BlankFlagged(flag*[Flag]*, redraw (optional)*[boolean]*)
- CentreOfGravity()
- Check(filename*[string]*, detailed (optional)*[boolean]*, json (optional)*[boolean]*, include (optional)*[boolean]*)
- ClearFlag(flag*[Flag]*)
- Copy(update (optional)*[boolean]*)
- CopyFlagged(flag*[Flag]*, update (optional)*[boolean]*)
- Delete()
- DeleteFlagged(flag*[Flag]*, recursive (optional)*[boolean]*)
- DeleteInclude(Include label*[integer]*, method (optional)*[constant]*, force (optional)*[boolean]*)
- FlagDuplicate(flag*[Flag]*)
- Flagged(flag*[Flag]*)
- GetIncludeTransformOffsets()
- Hide()
- Import(filename*[string]*)
- ImportInclude(source*[String OR Include Object]*, target (optional)*[Include Object]*)
- ImportIncludeTransform(filename*[string]*, idnoff*[integer]*, ideoff*[integer]*, idpoff*[integer]*, idmoff*[integer]*, idsoff*[integer]*, idfoff*[integer]*, iddoff*[integer]*, idroff*[integer]*)
- Mass()
- MassPropCalc(flag*[Flag]*)
- MergeNodes(flag*[Flag]*, dist*[real]*, label (optional)*[integer]*, position (optional)*[integer]*)
- PopulateInitialVelocities()
- PropagateFlag(flag*[Flag]*)
- RenumberAll(start*[integer]*)
- RenumberFlagged(flag*[Flag]*, start*[integer]*, mode (optional)*[constant]*)
- SetColour(colour*[colour from Colour class.]*)
- SetFlag(flag*[Flag]*)
- Show()
- Unblank()

- UnblankFlagged(flag*[Flag]*, redraw (optional)*[boolean]*)
- UnsketchAll(redraw (optional)*[boolean]*)
- UpdateGraphics()
- UsesLargeLabels()
- Write(filename*[string]*, options (optional)*[object]*)

Model constants

Constants for compress mode

Name	Description
Model.INDIVIDUAL_GZIP	Each file 'name.key' is 'gzipped' to become the individual file 'name.key.gz'
Model.INDIVIDUAL_ZIP	Each file 'name.key' is 'zipped' to become the individual file 'name.key.zip'
Model.KEEP_ORIGINAL	Each file 'name.key' is written using its original compression: uncompressed, '.gz. or '.zip' format
Model.PACKAGED_ZIP	Suitable for models with include files where the entire model is packed into a single .zip file, preserving its directory structure.

Constants for compress switch

Name	Description
Model.COMPRESS_KEEP	Keeps the keyout compression format same as that of what was read in.
Model.COMPRESS_OFF	Switches off compression during keyout.
Model.COMPRESS_ON	Switches on compression during keyout.

Constants for filetype

Name	Description
Model.ABAQUS	ABAQUS input file
Model.CAD	CAD file
Model.IGES	IGES 5.3 geometry file
Model.JT	JT geometry file
Model.LSDYNA	Ansys LS-DYNA keyword file
Model.NASTRAN	NASTRAN bulk data file
Model.RADIOSS	RADIOSS block format file
Model.STEP	STEP geometry file

Constants for include deletion

Name	Description
Model.REMOVE_FROM_SETS	Only deletes items within the include selected but may remove items from sets in other includes.
Model.REMOVE_INCLUDE_ONLY	Only deletes items within the include selected without removing items from sets in other includes.
Model.REMOVE_JUNIOR	Delete items in other includes if they 'belong' to items in this include file but are considered to be 'junior' in the standard PRIMER hierarchy.

Constants for mass_properties_calculation

Name	Description
Model.CENTRE_AT_COFG	Uses the centre at centre of gravity in calculation of inertia properties.
Model.GLOBAL_AXES	GLOBAL AXES
Model.LOCAL_AXES	LOCAL AXES
Model.PRINCIPAL_AXES	PRINCIPAL AXES
Model.USER_DEFINED_CENTRE	Uses the user defined centre in calculation of inertia properties.

Constants for merge

Name	Description
Model.DISCARD_PRIMARY_CLASH	Merge option - discard primary items only on clash
Model.DISCARD_SECONDARY_CLASH	Merge option - discard secondary items only on clash
Model.INCREASE_PRIMARY_ALWAYS	Merge option - increase primary items always
Model.INCREASE_PRIMARY_CLASH	Merge option - increase primary items only on clash
Model.INCREASE_SECONDARY_ALWAYS	Merge option - increase secondary items always
Model.INCREASE_SECONDARY_CLASH	Merge option - increase secondary items only on clash

Constants for renumber

Name	Description
Model.IGNORE_CLASH	Renumber option - Ignore clashes.
Model.MOVE_CLASH_UP	Renumber option - Move clashing > highest label.
Model.RENUMBER_TO_FREE	Renumber option - Renumber to next free label.
Model.SHIFT_ALL_UP	Renumber option - Shift upwards to make space.

Constants for write hook

Name	Description
Model.WRITE_DIALOGUE	Flag that triggers the write hook script from the dialogue box
Model.WRITE_INCLUDE_TREE	Flag that triggers the write hook script from the include file tree
Model.WRITE_MODEL	Flag that triggers the write hook script from the model tab
Model.WRITE_SELECT_INCLUDES	Flag that triggers the write hook script from the select include files panel

Model class properties

Name	Type	Description
highest (read only)	integer	The highest model number present in PRIMER

Model properties

Name	Type	Description
binary (read only)	boolean	If model is in binary then it will be 1(true) else 0(false).
comments	string	Comments stored at the top of the primary model file.

compress (read only)	boolean	If model is compressed then it will be 1(true) else 0(false).
compressMode (read only)	integer	This option can be used to know the mode of compression. Can be Model.INDIVIDUAL_GZIP or Model.INDIVIDUAL_ZIP or Model.PACKAGED_ZIP
control (read only)	Control object	Control cards for model. See Control for more details.
damping (read only)	Damping object	Damping cards for model. See Damping for more details.
database (read only)	Database object	Database cards for model. See Database for more details.
fileStartAscii (read only)	boolean	If the beginning of the master file (*CONTROL etc) is in ascii then 1(true) else 0(false)(NOTE: If master file is ascii then fileStartAscii won't be checked and show 0(false)).
filename (read only)	string	Name of file that model was read from (blank if model created)
id	logical	If ID flag set for *KEYWORD card
layer	integer	The current layer for the model. This is the label of the Include file or 0 for the main file. See also Include.MakeCurrentLayer()
loadBody (read only)	LoadBody object	LoadBody cards for model. See LoadBody for more details.
masterAscii (read only)	boolean	If master file is in ascii then 1(true) else 0(false).
num	string	Model num (for _ID)
number	integer	Model number
path (read only)	string	Path that model was read from (blank if model created)
project	string	Model project (for _ID)
readlog	string	Full path of the readlog file
stage	string	Model stage (for _ID)
title	string	Model title
visible	logical	Model visibility flag

Detailed Description

The Model class allows you to do various operations on models in PRIMER. There are various methods available that allow you do create, read, blank models etc. See the documentation below for more details.

Constructor

new Model(number (optional)*[integer]*)

Description

Create a new model in PRIMER

Arguments

- **number (optional)** (integer)

Model number to create. If omitted the next free model number will be used.

Returns

Model object

Return type

Model

Example

To create a new model

```
var m = new Model();
```

To create model 10

```
var m = new Model(10);
```

Details of functions

AreaVolumeFlagged(flag*[Flag]*)

Description

Calculates the Area/Volume of the selected items.
Note: The area calculation is based only on shell elements, and the volume calculation is based only on solid elements.

Arguments

- **flag** (Flag)

Flag set on entities you wish to calculate area/volume for

Returns

Object with the following properties:

Name	Type	Description
area	real	Area of flagged items
volume	real	Volume of flagged items

Return type

object

Example

To calculate the area/volume properties of the items flagged by flag f.

```
var props = m.AreaVolumeFlagged(f);
var area = props.area;
var volume = props.volume;
```

Attached(flag*[Flag]*, redraw (optional)*[boolean]*)

Description

Finds attached items to flagged items. The attached items are flagged with the same flag.

Arguments

- **flag** (Flag)

Flag set on items that you want to find attached to

- **redraw (optional)** (boolean)

If true, the display will be updated to display only the original flagged items and the attached items.

Returns

No return value

Example

To find items attached to items flagged with flag f in model m:

```
m.Attached(f);
```

Autofix()

Description

Autofix option does a model check and autofixes all the fixable errors in the model

Arguments

No arguments

Returns

No return value

Example

To autofix fixable errors of the model 'm'

```
m.Autofix();
```

Blank()

Description

Blanks a model in PRIMER

Arguments

No arguments

Returns

No return value

Example

To blank model object m

```
m.Blank();
```

BlankAll() [static]

Description

Blanks all models

Arguments

No arguments

Returns

No return value

Example

To blank all models

```
Model.BlankAll();
```

BlankFlagged(flag*[Flag]*, redraw (optional)*[boolean]*)

Description

Blanks all of the flagged items in the model.

Arguments

- **flag** (Flag)

Flag set on items that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank everything in model m flagged with flag f:

```
m.BlankFlagged(f);
```

CentreOfGravity()

Description

Returns the centre of gravity for a model

Arguments

No arguments

Returns

An array containing the x, y and z coordinates for the CofG.

Return type

Array

Example

To get the centre of gravity for model m:

```
var cofg = m.CentreOfGravity();
var x = cofg[0];
var y = cofg[1];
var z = cofg[2];
```

Check(filename*[string]*, detailed (optional)*[boolean]*, json (optional)*[boolean]*, include (optional)*[boolean]*)

Description

Checks a model, writing any errors to file.

Arguments

- **filename** (string)

Name of file to write errors to

- **detailed (optional)** (boolean)

If set to "true", detailed error messages are given.

- **json (optional)** (boolean)

If set, output in filename will be written in JSON format. If omitted json will be set to false. If JSON format is written then detailed will automatically be set. Note that when writing JSON format the labels produced can be strings instead of integers in some rare cases. If you are writing a script to read a JSON file, it must be able to cope with this. Specifically if the item is a character label the label will be a string. For child collect sets the label will be a string of the format 'X_Y' where X is the parent set label and Y will be the child set number (1, 2, 3 ...). In this case use Set.GetCollectChild() to get the object.

- **include (optional)** (boolean)

If set, error messages will be written in include by include layout. This option is not applicable if JSON is set.

Returns

No return value

Example

To check model m, writing detailed errors to file 'errors.txt' in include layout:

```
m.Check('errors.txt', true, false, true);
```

To check a model writing the warnings/errors as JSON to file 'errors.json', parse it and write them to the dialogue box:

```
m.Check('errors.json', true, true);
var f = new File('errors.json', File.READ);
var json = f.ReadAll();
f.Close();
var o = JSON.parse(json);
for (var e in o)                                    // "error" or "warning"
{
    Message(e);
    for (var t in o[e])                             // type
    {
        Message("  "+t);
        for (var m in o[e][t])                      // message
        {
            Message("    "+m);
            for (var i=0; i<o[e][t][m].length; i++)  // Array of objects
containing label and include
            {
                Message("        "+o[e][t][m][i].label+" (include
"+o[e][t][m][i].include+")");
            }
        }
    }
}
```

ClearFlag(flag*[Flag]*)

Description

Clears the flagging for a model in PRIMER. See also Model.PropagateFlag(), Model.SetFlag(), global.AllocateFlag() and global.ReturnFlag().

Arguments

- **flag** (Flag)

Flag to clear

Returns

No return value

Example

To clear flag f for everything in model m:

```
m.ClearFlag(f);
```

Copy(update (optional)*[boolean]*)

Description

Copy model to the next free model in PRIMER

Arguments

- **update (optional)** (boolean)

If the graphics should be updated after the model is copied. If omitted update will be set to false

Returns

Model object for new model.

Return type

Model

Example

To copy model m to the next free model in PRIMER.

```
var mnew = m.Copy();
```

CopyFlagged(flag*[Flag]*, update (optional)*[boolean]*)

Description

Copy flagged items in a model to the next free model in PRIMER

Arguments

- **flag** (Flag)

Flag set on items that you want to copy

- **update (optional)** (boolean)

If the graphics should be updated after the model is copied. If omitted update will be set to false

Returns

Model object for new model.

Return type

Model

Example

To copy everything in model m flagged with flag f to the next free model in PRIMER.

```
var mnew = m.CopyFlagged(f);
```

Delete()

Description

Deletes a model in PRIMER
Do not use the Model object after calling this method.

Arguments

No arguments

Returns

No return value

Example

To delete model m in PRIMER

```
m.Delete();
```

DeleteAll() [static]

Description

Deletes all existing models from PRIMER

Arguments

No arguments

Returns

No return value

Example

To delete all models

```
Model.DeleteAll();
```

DeleteFlagged(flag*[Flag]*, recursive (optional)*[boolean]*)

Description

Deletes all of the flagged items in the model. Note that this may not actually delete all of the items. For example if a node is flagged but the node is used in a shell which is not flagged then the node will not be deleted.

Arguments

- **flag** (Flag)

Flag set on items that you want to delete

- **recursive (optional)** (boolean)

If deletion is recursive (for example, if a shell is deleted with recursion on the shell nodes will be deleted if possible). If omitted recursive will be set to true.

Returns

No return value

Example

To delete everything in model m flagged with flag f:

```
m.DeleteFlagged(f);
```

DeleteInclude(Include label*[integer]*, method (optional)*[constant]*, force (optional)*[boolean]*)

Description

Tries to delete an include file from the model. Note that this may not actually delete the include file. For example if some of the items in the include file are required by other things in different includes then the include file will not be deleted.

Arguments

- **Include label** (integer)

label of include file that you want to delete

- **method (optional)** (constant)

Method for deleting items. Must be Model.REMOVE_FROM_SETS (default), Model.REMOVE_JUNIOR or Model.REMOVE_INCLUDE_ONLY.
Model.REMOVE_FROM_SETS will only delete items within the include selected but may remove items from sets in other includes.
Model.REMOVE_JUNIOR may delete items in other includes - this will happen if they 'belong' to items in this include and are considered 'junior'
Model.REMOVE_INCLUDE_ONLY does the same as Model.REMOVE_FROM_SETS but will **not** remove items from sets in other includes.

- **force (optional)** (boolean)

Forcible deletion option (for example, a node is deleted even when it is referenced by a shell which is not deleted). This will remove the include file (not just the contents) from the model. If this argument is omitted, force will be set to false.

Returns

true if include successfully deleted, false otherwise

Return type

Boolean

Example

To delete include file number 5 in model m removing items from sets in other includes if required:

```
m.DeleteInclude(5, 1);
```

First() [static]

Description

Returns the Model object for the first model in PRIMER (or null if there are no models)

Arguments

No arguments

Returns

Model object

Return type

Model

Example

To get the Model object for the first model:

```
var m = Model.First();
```

FirstFreeItemLabel(type*[string]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free label for an item type in the model. Also see Model.LastFreeItemLabel() and Model.NextFreeItemLabel().

Arguments

- **type** (string)

The type of the item (for a list of types see Appendix I of the PRIMER manual).

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

integer

Return type

Number

Example

To get the first free node label in model m:

```
var label = m.FirstFreeItemLabel("NODE");
```

FlagDuplicate(flag*[Flag]*)

Description

Flag all nodes referenced in two different includes

Arguments

- **flag** (Flag)

Flag which will be used to flag the "duplicate" nodes

Returns

No return value

Example

To Flag with flag f all the nodes referenced in different includes from model m

```
m.FlagDuplicate(f);
```

Flagged(flag*[Flag]*)

Description

Checks if the model is flagged or not

Arguments

- **flag** (Flag)

Flag to test on the model

Returns

true if flagged, false if not

Return type

Boolean

Example

To check if model m has flag f set on it:

```
if (m.Flagged(f) ) do_something...
```

GetAll() [static]

Description

Returns an array of Model objects for all the models in PRIMER

Arguments

No arguments

Returns

Array of Model objects

Return type

Array

Example

To make an array of Model objects for all of the models in PRIMER

```
var m = Model.GetAll();
```

GetFromID(model number*[integer]*) [static]

Description

Returns the Model object for a model ID or null if model does not exist

Arguments

- **model number** (integer)

number of the model you want the Model object for

Returns

Model object

Return type

Model

Example

To get the Model object for model number 1

```
var m = Model.GetFromID(1);
```

GetIncludeTransformOffsets()

Description

Looks at all of the items in the model and determines values for IDNOFF, IDEOFF, IDPOFF etc that could be used with Model.ImportIncludeTransform to guarantee that there would not be any clashes with existing items in the model.

Arguments

No arguments

Returns

Object with the following properties:

Name	Type	Description
iddoff	integer	Offset to define id
ideoff	integer	Offset to element id
idfoff	integer	Offset to function id
idmoff	integer	Offset to material id
idnoff	integer	Offset to node id
idpoff	integer	Offset to part id
idroff	integer	Offset to section, hourglass, EOS id
idsoff	integer	Offset to set id

Return type

object

Example

To determine offsets for model m and then import an include transform "test.inc":

```
var o = m.GetIncludeTransformOffsets();
if (o)
{
    var success = m.ImportIncludeTransform("test.inc", o.idnoff, o.ideoff,
o.idpoff, o.idmoff, o.idsoff, o.idfoff, o.iddoff, o.idroff);
}
```

Hide()

Description

Hides a model in PRIMER

Arguments

No arguments

Returns

No return value

Example

To hide model m in PRIMER

```
m.Hide();
```

Import(filename*[string]*)

Description

Imports a file into model m. The model can already contain items. However, **note that if the file cannot be imported because of a label clash or other problem PRIMER may delete the model and the script will terminate**. Note prior to v17 of PRIMER imported data would always be imported to the master model, irrespective of the current layer. From v17 onwards this has been corrected and the current layer is used to determine the destination of imported data.

Arguments

- **filename** (string)

Filename of the Ansys LS-DYNA keyword file you want to import

Returns

0: No errors/warnings.
> 0: This number of errors occurred.
< 0: Absolute number is the number of warnings that occurred.

Return type

Number

Example

To import file "test.key" into model m

```
m.Import("test.key");
```

ImportInclude(source*[String OR Include Object]*, target (optional)*[Include Object]*)

Description

Imports a keyword file or an Include object from different model as a **new include or into an existing include file** for model m. The labels of any items in the imported include contents that clash with existing labels will automatically be renumbered with one exception. The behaviour for *SET_COLLECT cards can be controlled with Options.merge_set_collect.

Arguments

- **source** (String OR Include Object)

Can either be a Filename of the Ansys LS-DYNA include file you want to import, OR Include object of another model you want to import

- **target (optional)** (Include Object)

Include file object of current model if the Import has to be done in an existing include.
If not using this argument the contents of the source will be imported as a new include.
If using this argument the contents of the source will NOT be imported as a new include, they will be merged with the target include.
Note: Target cannot be include number 0 (it must be an include file, not the master file)

Returns

Include object for include file

Return type

Include

Example

To import include file "include.key" into model m (include.key will be a new include file in model m):

```
m.ImportInclude("include.key");
```

To import include file "include.key" into existing include number 3 in model m (include.key will be merged with include number 3 of m):

```
var incl_target = Include.GetFromID(m, 3);
m.ImportInclude("include.key", incl_target);
```

To import include number 5 from model m2 into model m1 (include number 5 from m2 will be a new include file in model m1):

```
var incl_source = Include.GetFromID(m2, 5);
m1.ImportInclude(incl_source);
```

To import include number 5 from model m2 into existing include number 3 in model m1 (include number 5 from m2 will be merged with include number 3 in m1):

```
var incl_source = Include.GetFromID(m2, 5);
var incl_target = Include.GetFromID(m1, 3);
m1.ImportInclude(incl_source, incl_target);
```

ImportIncludeTransform(filename*[string]*, idnoff*[integer]*, ideoff*[integer]*, idpoff*[integer]*, idmoff*[integer]*, idsoff*[integer]*, idfoff*[integer]*, iddoff*[integer]*, idroff*[integer]*)

Description

Imports a file as an include transform file for model m. The labels of any items in the include file will be renumbered by idnoff, ideoff etc.

Arguments

- **filename** (string)

Filename of the Ansys LS-DYNA include file you want to import

- **idnoff** (integer)

Offset for nodes in the file

- **ideoff** (integer)

Offset for elements in the file

- **idpoff** (integer)

Offset for parts in the file

- **idmoff** (integer)

Offset for materials in the file

- **idsoff** (integer)

Offset for sets in the file

- **idfoff** (integer)

Offset for functions and tables in the file

- **iddoff** (integer)

Offset for defines in the file

- **idroff** (integer)

Offset for other labels in the file

Returns

Include object if successful, null if not

Return type

Include

Example

To import include transform file "include.key" into model m using 1000 for all offsets

```
m.ImportIncludeTransform("include.key", 1000, 1000, 1000, 1000, 1000, 1000,
1000, 1000);
```

Last() [static]

Description

Returns the Model object for the last model in PRIMER (or null if there are no models)

Arguments

No arguments

Returns

Model object

Return type

Model

Example

To get the Model object for the last model:

```
var m = Model.Last();
```

LastFreeItemLabel(type*[string]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free label for an item type in the model. Also see Model.FirstFreeItemLabel() and Model.NextFreeItemLabel().

Arguments

- **type** (string)

The type of the item (for a list of types see Appendix I of the PRIMER manual).

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

integer

Return type

Number

Example

To get the last free node label in model m:

```
var label = m.LastFreeItemLabel("NODE");
```

Mass()

Description

Returns the mass for a model

Arguments

No arguments

Returns

real

Return type

Number

Example

To get the mass for model m:

```
var mass = m.Mass();
```

MassPropCalc(flag*[Flag]*)

Description

Calculates the Mass, CoG, and Intertia Tensor of the flagged items and returns an object with the above properties. See Properties for mass properties calculation under options class to configure inclusion of lumped mass, etc.

Arguments

- **flag** (Flag)

Calculate mass propetries of flagged items

Returns

Object with the following properties:

Name	Type	Description
cofgx	real	X coordinate of centre of gravity
cofgy	real	Y coordinate of centre of gravity
cofgz	real	Z coordinate of centre of gravity
inerxx	real	XX inertia
inerxy	real	XY inertia
inerxz	real	XZ inertia
ineryy	real	YY inertia
ineryz	real	YZ inertia
inerzz	real	ZZ inertia
mass	real	Mass

Return type

object

Example

To calculate the mass properties of the items flagged by flag f

```
var props = m.MassPropCalc(f);
```

Merge(Primary Model*[Model]*, Secondary Model*[Model]*, Option to fix clashes (optional)*[constant]*, Merge nodes flag (optional)*[boolean]*, dist (required if merge nodes flag used) (optional)*[real]*, label (optional)*[integer]*, position (optional)*[integer]*) [static]

Description

Merge 2 models together to make a new model.

Arguments

- **Primary Model** (Model)

Primary Model for merge.

- **Secondary Model** (Model)

Secondary Model for merge.

- **Option to fix clashes (optional)** (constant)

Type of fix. Can be Model.INCREASE_SECONDARY_ALWAYS, Model.INCREASE_SECONDARY_CLASH, Model.DISCARD_SECONDARY_CLASH, Model.INCREASE_PRIMARY_ALWAYS, Model.INCREASE_PRIMARY_CLASH or Model.DISCARD_PRIMARY_CLASH

- **Merge nodes flag (optional)** (boolean)

If this flag is set to true, PRIMER will merge nodes after the model merge.

- **dist (required if merge nodes flag used) (optional)** (real)

Nodes closer than dist will be potentially merged.

- **label (optional)** (integer)

Label to keep after merge. If > 0 then highest label kept. If <= 0 then lowest kept. If omitted the lowest label will be kept.

- **position (optional)** (integer)

Position to merge at. If > 0 then merged at highest label position. If < 0 then merged at lowest label position. If 0 then merged at midpoint. If omitted the merge will be done at the lowest label.

Returns

Model object (or null if the merge is unsuccessful)

Return type

Model

Example

To merge models m1 and m2 together:

```
var m = Model.Merge(m1, m2);
```

MergeNodes(flag*[Flag]*, dist*[real]*, label (optional)*[integer]*, position (optional)*[integer]*)

Description

Attempts to merge nodes on items flagged with flag for this model in PRIMER. Merging nodes on *AIRBAG_SHELL_REFERENCE_GEOMETRY can be controlled by using Options.node_replace_asrg. Also see Node.Merge().

Arguments

- **flag** (Flag)

Flag set on items to merge nodes

- **dist** (real)

Nodes closer than dist will be potentially merged.

- **label (optional)** (integer)

Label to keep after merge. If > 0 then highest label kept. If <= 0 then lowest kept. If omitted the lowest label will be kept.

- **position (optional)** (integer)

Position to merge at. If > 0 then merged at highest label position. If < 0 then merged at lowest label position. If 0 then merged at midpoint. If omitted the merge will be done at the lowest label.

Returns

The number of nodes merged

Return type

Number

Example

To (try to) merge nodes on everything in model m flagged with flag f, with a distance of 0.1:

```
m.MergeNodes(f, 0.1);
```

NextFreeItemLabel(type*[string]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free label for an item type in the model. Also see Model.FirstFreeItemLabel() and Model.LastFreeItemLabel().

Arguments

- **type** (string)

The type of the item (for a list of types see Appendix I of the PRIMER manual).

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

integer

Return type

Number

Example

To get the next free node label in model m:

```
var label = m.NextFreeItemLabel("NODE");
```

PopulateInitialVelocities()

Description

Populate the initial velocity field (nvels) for all nodes of the model

Arguments

No arguments

Returns

No return value

Example

```
m.PopulateInitialVelocities();
```

PropagateFlag(flag*[Flag]*)

Description

Propagates the flagging for a model in PRIMER. For example if a part in the model is flagged, this will flag the elements in the part, and the nodes on those elements. However, the propagation is not just for elements and nodes. Expanding on the above example, if the part is used in a *CONSTRAINED_EXTRA_NODES definition, that will be flagged, and the nodes that refers to will be flagged. PRIMER will use all of the keywords in the model and use them appropriately.
See also Model.ClearFlag(), Model.SetFlag(), global.AllocateFlag() and global.ReturnFlag().

Arguments

- **flag** (Flag)

Flag to propagate

Returns

No return value

Example

To propagate the flagging in model m for flag f

```
m.PropagateFlag(f);
```

Read(filename*[string]*, filetype (optional)*[constant]*, number (optional)*[integer]*) [static]

Description

Reads a file into the first free model in PRIMER

Arguments

- **filename** (string)

Filename you want to read

- **filetype (optional)** (constant)

Filetype you want to read. Can be Model.LSDYNA, Model.ABAQUS, Model.NASTRAN, Model.RADIOSS, Model.CAD, Model.IGES, Model.JT, or Model.STEP. If omitted the file is assumed to be a DYNA3D file. For Model.NASTRAN there are options that change how the model is read. See Options for details.

- **number (optional)** (integer)

Model number to read file into. If omitted the next free model number will be used.

Returns

Model object (or null if error)

Return type

Model

Example

To read the keyword file /data/test/file.key

```
Model.Read("/data/test/file.key");
```

To read the NASTRAN file /data/test/file.dat

```
Model.Read("/data/test/file.dat", Model.NASTRAN);
```

To read the keyword file /data/test/file.key into model 10

```
Model.Read("/data/test/file.key", Model.LSDYNA, 10);
```

RenumberAll(start*[integer]*)

Description

Renumbers all of the items in the model.

Arguments

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber everything in model m, starting at 1000000:

```
m.RenumberAll(1000000);
```

RenumberFlagged(flag*[Flag]*, start*[integer]*, mode (optional)*[constant]*)

Description

Renumbers all of the flagged items in the model.

Arguments

- **flag** (Flag)

Flag set on items that you want to renumber

- **start** (integer)

Start point for renumbering

- **mode (optional)** (constant)

Renumber mode. Can be Model.IGNORE_CLASH, Model.MOVE_CLASH_UP, Model.SHIFT_ALL_UP, or Model.RENUMBER_TO_FREE (default),

Returns

No return value

Example

To renumber everything in model m flagged with flag f, starting at 1000000, using mode MOVE_CLASH_UP:

```
m.RenumberFlagged(f, 1000000, Model.MOVE_CLASH_UP);
```

Select(prompt*[string]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select a model using standard PRIMER object menus. If there are no models in memory then Select returns null. If only one model is present then the model object is returned. If there is more than one model in memory then an object menu is mapped allowing the user to choose a model.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Model object (or null if no models present).

Return type

Model

Example

To select a model giving the prompt 'Select model':

```
var m = Model.Select('Select model');
```

SetColour(colour*[colour from Colour class.]*)

Description

Sets the colour of the model.

Arguments

- **colour** (colour from Colour class.)

The colour you want to set the model to

Returns

No return value

Example

To set the colour of model m to red:

```
m.SetColour(Colour.RED);
```

or

```
m.SetColour(Colour.RGB(255, 0, 0));
```

SetFlag(flag*[Flag]*)

Description

Sets the flagging for a model in PRIMER. See also Model.PropagateFlag(), Model.ClearFlag(), global.AllocateFlag() and global.ReturnFlag().

Arguments

- **flag** (Flag)

Flag to set

Returns

No return value

Example

To set flag f for everything in model m:

```
m.SetFlag(f);
```

Show()

Description

Shows a model in PRIMER

Arguments

No arguments

Returns

No return value

Example

To show model m in PRIMER

```
m.Show();
```

Total() [static]

Description

Returns the total number of models.

Arguments

No arguments

Returns

integer

Return type

Number

Example

To find how many models there are in PRIMER:

```
var num = Model.Total();
```

Unblank()

Description

Unblanks a model in PRIMER

Arguments

No arguments

Returns

No return value

Example

To unblank model m

```
m.Unblank();
```

UnblankAll() [static]

Description

Unblanks all models

Arguments

No arguments

Returns

No return value

Example

To unblank all models

```
Model.UnblankAll();
```

UnblankFlagged(flag*[Flag]*, redraw (optional)*[boolean]*)

Description

Unblanks all of the flagged items in the model.

Arguments

- **flag** (Flag)

Flag set on items that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank everything in model m flagged with flag f:

```
m.UnblankFlagged(f);
```

UnsketchAll(redraw (optional)*[boolean]*)

Description

Unsketches all of the sketched items in the model.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the items are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all the sketched objects in model m:

```
m.UnsketchAll();
```

UpdateGraphics()

Description

Updates the graphics for a model in PRIMER

Arguments

No arguments

Returns

No return value

Example

To update the graphics for model m

```
m.UpdateGraphics();
```

UsesLargeLabels()

Description

Checks to see if a model uses large labels

Arguments

No arguments

Returns

logical, true if model uses large labels, false otherwise

Return type

Boolean

Example

To check if model m uses large labels:

```
var large = m.UsesLargeLabels();
```

Write(filename*[string]*, options (optional)*[object]*)

Description

Writes a model in PRIMER to file

Arguments

- **filename** (string)

Filename of the Ansys LS-DYNA keyword file you want to write

- **options (optional)** (object)

Options specifying how the file should be written out. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
binary (optional)	boolean	If true then the entire output file will be written out in binary, if false then the entire file will be written in ascii. If not defined the default is for each file in the model to be written in its original format, or if this is a newly created model ascii will be used.
compress (optional)	boolean	If true then the output file will be compressed. If false (default) then an uncompressed file will be written.
compressLevel (optional)	integer	Compression level for .gz and .zip files. Must be in the range 1 to 9 with 1 being the least compression (fastest speed) to 9 being the greatest compression (slowest speed)
compressMode (optional)	integer	This option can be used to specify the mode of compression. Can be Include.KEEP_ORIGINAL or Include.INDIVIDUAL_GZIP or Include.INDIVIDUAL_ZIP
fileStartAscii (optional)	boolean	Only relevant if binary output format has been selected. If true then the beginning of the file (*CONTROL etc) file is written out in ascii. If false (default) then the entire file is converted to binary.
i10 (optional)	boolean	If true then i10 format will be used to write the file. If false (default) then the original Ansys LS-DYNA format in which the file was written will be used, or if this is a newly created model the smallest format which will contain it will be used. Note that large format is only available from version R9 and above. See also the large property.
kbyExt (optional)	boolean	If true then a binary format output file will be given the extension .kby, replacing the existing extension.
large (optional)	boolean	If true then large format will be used to write the file. If false (default) then the normal Ansys LS-DYNA format will be used. Note that large format is only available from version R7.1 and above.
masterAscii (optional)	boolean	Only relevant if binary output format has been selected. If true then the whole master file is written out in ascii. If false (default) then the master file is also converted to binary.
method (optional)	integer	The method used to write include files. Can be Include.MASTER_ONLY, Include.MERGE, Include.NOT_WRITTEN, Include.SUBDIR (default) or Include.SAME_DIR
parametersAsValues (optional)	boolean	If true then the underlying values of any parameters will be written when they are used in data fields rather than '&name'. If false then '&name' will be written. See also Options.keyout_parameter_values
path (optional)	integer	The method used to write include paths. Can be Include.ABSOLUTE (default) or Include.RELATIVE
separator (optional)	integer	The directory separator used when writing include files. Can be Include.NATIVE (default), Include.UNIX or Include.WINDOWS
version (optional)	string	The Ansys LS-DYNA version used to write the file. Can be "971R5", "971R4", "971R3", "970v6763" etc (see the version popup in Model->Write '>>> Ansys LS-DYNA output options' for a full list). See also Options.dyna_version

Returns

No return value

Example

To Write model m to file /data/test/file.key as a compressed gzip in version R10.0

```
var output_obj = new Object();
output_obj.version = "R10.0";
output_obj.compress = true;
output_obj.compressMode = Model.INDIVIDUAL_GZIP;
m.Write("/data/test/file.key", output_obj);
```

MorphBox class

The MorphBox class gives you access to morph boxes in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model *[Model]*, redraw (optional) *[boolean]*)
- BlankFlagged(Model *[Model]*, flag *[Flag]*, redraw (optional) *[boolean]*)
- First(Model *[Model]*)
- FirstFreeLabel(Model *[Model]*, layer (optional) *[Include number]*)
- FlagAll(Model *[Model]*, flag *[Flag]*)
- FlagAllMorphedConnections(model *[Model]*, flag *[integer]*)
- ForEach(Model *[Model]*, func *[function]*, extra (optional) *[any]*)
- GetAll(Model *[Model]*, property (optional) *[string]*)
- GetFlagged(Model *[Model]*, flag *[Flag]*, property (optional) *[string]*)
- GetFromID(Model *[Model]*, number *[integer]*)
- Last(Model *[Model]*)
- LastFreeLabel(Model *[Model]*, layer (optional) *[Include number]*)
- NextFreeLabel(Model *[Model]*, layer (optional) *[Include number]*)
- Pick(prompt *[string]*, limit (optional) *[Model or Flag]*, modal (optional) *[boolean]*, button text (optional) *[string]*)
- RenumberAll(Model *[Model]*, start *[integer]*)
- RenumberFlagged(Model *[Model]*, flag *[Flag]*, start *[integer]*)
- Select(flag *[Flag]*, prompt *[string]*, limit (optional) *[Model or Flag]*, modal (optional) *[boolean]*)
- SetMorphConnections(status *[boolean]*)
- SketchFlagged(Model *[Model]*, flag *[Flag]*, redraw (optional) *[boolean]*)
- Total(Model *[Model]*, exists (optional) *[boolean]*)
- UnblankAll(Model *[Model]*, redraw (optional) *[boolean]*)
- UnblankFlagged(Model *[Model]*, flag *[Flag]*, redraw (optional) *[boolean]*)
- UnflagAll(Model *[Model]*, flag *[Flag]*)
- UnsketchAll(Model *[Model]*, redraw (optional) *[boolean]*)
- UnsketchFlagged(Model *[Model]*, flag *[Flag]*, redraw (optional) *[boolean]*)

Member functions

- ApplyMorphing(redraw (optional) *[boolean]*)
- AssociateComment(Comment *[Comment]*)
- Blank()
- Blanked()
- ClearFlag(flag *[Flag]*)
- Copy(range (optional) *[boolean]*)
- DetachComment(Comment *[Comment]*)
- Error(message *[string]*, options (optional) *[object]*)
- Error(message *[string]*, details (optional) *[string]*) [deprecated]
- FlagMorphedConnections(flag *[integer]*)
- Flagged(flag *[Flag]*)
- GetComments()
- GetParameter(prop *[string]*)
- GetPoint(xindex *[integer]*, yindex *[integer]*, zindex *[integer]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- Reset(redraw (optional) *[boolean]*)
- SetFlag(flag *[Flag]*)
- SetPointID(xindex *[integer]*, yindex *[integer]*, zindex *[integer]*, id *[integer]*)
- Sketch(redraw (optional) *[boolean]*)
- Unblank()

- Unsketch(redraw (optional)*[boolean]*)
- UpdateParametricCoordinates()
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

MorphBox properties

Name	Type	Description
exists (read only)	logical	true if box exists, false if referred to but not defined.
include	integer	The Include file number that the box is in.
label	integer	MorphBox number.
model (read only)	integer	The Model number that the box is in.
nx (read only)	integer	Number of morph points in parametric X direction
ny (read only)	integer	Number of morph points in parametric Y direction
nz (read only)	integer	Number of morph points in parametric Z direction
setid (read only)	integer	ID for node set of nodes dragged with the box. This will be a *SET_NODE_COLUMN containing the nodes together with their parametric coordinates in X, Y, Z. It is strongly discouraged to edit the contents of this set or the column data manually.

Detailed Description

The MorphBox class allows you to create, modify and manipulate morph boxes. See the documentation below for more details.

Constructor

new MorphBox(Model*[Model]*, label*[integer]*, flag*[Flag]*, options (optional)*[object]*)

Description

Create a new MorphBox object around flagged items.

Arguments

- **Model** (Model)

Model that morph box will be created in

- **label** (integer)

MorphBox number

- **flag** (Flag)

Flag set on the entities (for example nodes, elements and/or parts) that you want to create the box around

- **options (optional)** (object)

Options to create the box.

Object has the following properties:

Name	Type	Description

csys (optional)	integer	Coordinate system for local coordinates. Leave undefined if using global coordinates or if local coordinate system defined with n1, n2 and n3.
n1 (optional)	integer	Node 1 label for local coordinate. Leave undefined if using global coordinates or if local coordinate system defined with csys.
n2 (optional)	integer	Node 2 label for local coordinate. Leave undefined if using global coordinates or if local coordinate system defined with csys.
n3 (optional)	integer	Node 3 label for local coordinate. Leave undefined if using global coordinates or if local coordinate system defined with csys.
nx (optional)	integer	Number of points in X direction of box (assumed to be 2 for linear box if omitted)
ny (optional)	integer	Number of points in Y direction of box (assumed to be 2 for linear box if omitted)
nz (optional)	integer	Number of points in Z direction of box (assumed to be 2 for linear box if omitted)
points (optional)	Array of integers	Array of integers of depth 3 containing the morph point IDs. This should be omitted in the (default) case of also creating new morph points together with the morph box at the locations based on the bounding box of the flagged items. If this array contains ’nx’ by ’ny’ by ’nz’ existing morph points, the morph box is attached to these points, and ’csys’, ’n1’, ’n2’, ’n3’ will be irrelevant. Each ’points[i][j][k]’ should contain the morph point ID to be added at index i in local X direction, index j in local Y direction and index k in local Z direction. The box will then still contain flagged nodes only, but nodes geometrically outside the volume of the morph points will not be included either.

Returns

MorphBox object

Return type

MorphBox

Example

To create a new morph box in model m with label 100 and 2 by 2 by 2 points (linear in each coordinate direction) around all items flagged with flag in global coordinates:

```
var box = new MorphBox(m, 100, flag);
```

To create a new morph box in model m with label 100 and 4 by 4 by 2 points (cubic in parametric X and Y directions and linear in Z direction) around all flagged items in local coordinates determined by nodes 11, 12 and 13:

```
var options = new Object();
options.nx = 4;
options.ny = 4;
options.nz = 2;
options.n1 = 11;
options.n2 = 12;
options.n3 = 13;
var box = new MorphBox(m, 100, flag, options);
```

Suppose there are already morph points 1, 2, 3, 4, 5, 6, 7, 8 in model m at coordinates (0, 0, 0), (0, 0, 100), (0, 100, 0), (0, 100, 100), (100, 0, 0), (100, 0, 100), (100, 100, 0), (100, 100, 100) respectively. To create a new linear morph box between these points containing flagged items inside their volume:

```
var options = new Object();
options.points = [[[1,2],[3,4]],[[5,6],[7,8]]];
var box = new MorphBox(m, 100, flag, options);
```

Details of functions

ApplyMorphing(redraw (optional)*[boolean]*)

Description

Recalculates the X, Y and Z coordinates of all nodes linked to the morph box by the *SET_NODE_COLUMN. This should be called when coordinates of morph points have changed and you wish to apply the morphing. If several morph point positions on the same box change, then it is more speed-efficient to call this function only once for the box.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to apply the morphing to several boxes and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using Model.UpdateGraphics().

Returns

No return value

Example

To calculate all global X, Y and Z coordinates for the morphed nodes for box b:

```
b.ApplyMorphing();
```

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a box.

Arguments

- **Comment** (Comment)

Comment that will be attached to the box

Returns

No return value

Example

To associate comment c to the box b:

```
b.AssociateComment(c);
```

Blank()

Description

Blanks the box

Arguments

No arguments

Returns

No return value

Example

To blank box b:

```
b.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the boxs in the model.

Arguments

- **Model** (Model)

Model that all boxs will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the boxs in model m:

```
MorphBox.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged boxs in the model.

Arguments

- **Model** (Model)

Model that all the flagged boxs will be blanked in

- **flag** (Flag)

Flag set on the boxs that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the boxs in model m flagged with f:

```
MorphBox.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the box is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if box b is blanked:

```
if (b.Blanked() ) do_something...
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the box.

Arguments

- **flag** (Flag)

Flag to clear on the box

Returns

No return value

Example

To clear flag f for box b:

```
b.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the box. The target include of the copied box can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

MorphBox object

Return type

MorphBox

Example

To copy box b into box z:

```
var z = b.Copy();
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a box.

Arguments

- **Comment** (Comment)

Comment that will be detached from the box

Returns

No return value

Example

To detach comment c from the box b:

```
b.DetachComment(c);
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for box with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The box object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for box b:

```
b.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for box b:

```
b.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for box. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for box b:

```
b.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first box in the model.

Arguments

- **Model** (Model)

Model to get first box in

Returns

MorphBox object (or null if there are no boxs in the model).

Return type

MorphBox

Example

To get the first box in model m:

```
var b = MorphBox.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free box label in the model. Also see MorphBox.LastFreeLabel(), MorphBox.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free box label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

MorphBox label.

Return type

Number

Example

To get the first free box label in model m:

```
var label = MorphBox.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the boxs in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all boxs will be flagged in

- **flag** (Flag)

Flag to set on the boxs

Returns

No return value

Example

To flag all of the boxs with flag f in model m:

```
MorphBox.FlagAll(m, f);
```

FlagAllMorphedConnections(model*[Model]*, flag*[integer]*) [static]

Description

Flags all connections, in a given model, that have been morphed since their last remake. This includes connections that have been morphed by a morph box that has since been deleted.

Arguments

- **model** (Model)

Model containing desired connections.

- **flag** (integer)

Flag to mark morphed connections.

Returns

true if successful, false if not.

Return type

Boolean

Example

To flag all morphed connections in Model m with flag.

```
var flag = AllocateFlag();
   MorphBox.FlagAllMorphedConnections(m, flag);
```

FlagMorphedConnections(flag*[integer]*)

Description

Flags all connections that have been morphed, by a givine morph box, since their last remake. A connection could be morphed by one morph box and not another, therefore calling this function on two boxes that share a connection may produce different results depending on which box the function is called for. E.g. morb1 and morb2 share conx1, morb1 gets morphed whereas morb2 remains unchanged. Calling this function for morb1 will flag conx1, however calling the function for morb2 won't flag conx1.

Arguments

- **flag** (integer)

Flag to mark morphed connections.

Returns

true if successful, false if not.

Return type

Boolean

Example

To flag all morphed connections in a MorphBox with flag.

```
var flag = AllocateFlag();
   box.FlagMorphedConnections(flag);
```

Flagged(flag*[Flag]*)

Description

Checks if the box is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the box

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if box b has flag f set on it:

```
if (b.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each box in the model.
Note that ForEach has been designed to make looping over boxs as fast as possible and so has some limitations. Firstly, a single temporary MorphBox object is created and on each function call it is updated with the current box data. This means that you should not try to store the MorphBox object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new boxs inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all boxs are in

- **func** (function)

Function to call for each box

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the boxs in model m:

```
MorphBox.ForEach(m, test);
function test(b)
{
// b is MorphBox object
}
```

To call function test for all of the boxs in model m with optional object:

```
var data = { x:0, y:0 };
MorphBox.ForEach(m, test, data);
function test(b, extra)
{
// b is MorphBox object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of MorphBox objects or properties for all of the boxs in a model in PRIMER. If the optional property argument is not given then an array of MorphBox objects is returned. If the property argument is given, that property value for each box is returned in the array instead of a MorphBox object

Arguments

- **Model** (Model)

Model to get boxs from

- **property (optional)** (string)

Name for property to get for all boxs in the model

Returns

Array of MorphBox objects or properties

Return type

Array

Example

To make an array of MorphBox objects for all of the boxs in model m:

```
var a = MorphBox.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each box in model m:

```
var a = MorphBox.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a box.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the box b:

```
var comm_array = b.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of MorphBox objects for all of the flagged boxs in a model in PRIMER If the optional property argument is not given then an array of MorphBox objects is returned. If the property argument is given, then that property value for each box is returned in the array instead of a MorphBox object

Arguments

- **Model** (Model)

Model to get boxs from

- **flag** (Flag)

Flag set on the boxs that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged boxs in the model

Returns

Array of MorphBox objects or properties

Return type

Array

Example

To make an array of MorphBox objects for all of the boxs in model m flagged with f:

```
var b = MorphBox.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the boxs in model m flagged with f:

```
var a = MorphBox.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the MorphBox object for a box ID.

Arguments

- **Model** (Model)

Model to find the box in

- **number** (integer)

number of the box you want the MorphBox object for

Returns

MorphBox object (or null if box does not exist).

Return type

MorphBox

Example

To get the MorphBox object for box 100 in model m

```
var b = MorphBox.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a MorphBox property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the MorphBox.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

box property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if MorphBox property b.example is a parameter:

```
Options.property_parameter_names = true;
if (b.GetParameter(b.example) ) do_something...
Options.property_parameter_names = false;
```

To check if MorphBox property b.example is a parameter by using the GetParameter method:

```
if (b.ViewParameters().GetParameter(b.example) ) do_something...
```

GetPoint(xindex*[integer]*, yindex*[integer]*, zindex*[integer]*)

Description

Returns the morph point ID on the morph box at indices in X, Y and Z directions.

Arguments

- **xindex** (integer)

Index of the point in X direction. Note that indices start at 0, so it should be 0 for the points with the smallest parameteric X coordinate and box.nx-1 for the points with the highest X.

- **yindex** (integer)

Index of the point in Y direction. Note that indices start at 0, so it should be 0 for the points with the smallest parameteric Y coordinate and box.ny-1 for the points with the highest Y.

- **zindex** (integer)

Index of the point in Z direction. Note that indices start at 0, so it should be 0 for the points with the smallest parameteric Z coordinate and box.nz-1 for the points with the highest Z.

Returns

A MorphPoint object for the point on the box at given indices.

Return type

MorphPoint

Example

To get the 2nd point on the edge along the local Y direction and at highest local X and lowest local Z coordinate:

```
var point = box.GetPoint(box.nx-1, 1, 0);
```

Keyword()

Description

Returns the keyword for this morph box (*MORPH_BOX or *MORPH_BOX_HIGH_ORDER). **Note that a carriage return is not added**. See also MorphBox.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for morph box b:

```
var key = b.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the morph box. **Note that a carriage return is not added**. See also MorphBox.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for morph box b:

```
var cards = b.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last box in the model.

Arguments

- **Model** (Model)

Model to get last box in

Returns

MorphBox object (or null if there are no boxs in the model).

Return type

MorphBox

Example

To get the last box in model m:

```
var b = MorphBox.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free box label in the model. Also see MorphBox.FirstFreeLabel(), MorphBox.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free box label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

MorphBox label.

Return type

Number

Example

To get the last free box label in model m:

```
var label = MorphBox.LastFreeLabel(m);
```

Next()

Description

Returns the next box in the model.

Arguments

No arguments

Returns

MorphBox object (or null if there are no more boxs in the model).

Return type

MorphBox

Example

To get the box in model m after box b:

```
var b = b.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) box label in the model. Also see MorphBox.FirstFreeLabel(), MorphBox.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free box label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

MorphBox label.

Return type

Number

Example

To get the next free box label in model m:

```
var label = MorphBox.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a box.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only boxs from that model can be picked. If the argument is a Flag then only boxs that are flagged with *limit* can be selected. If omitted, or null, any boxs from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

MorphBox object (or null if not picked)

Return type

MorphBox

Example

To pick a box from model m giving the prompt 'Pick box from screen':

```
var b = MorphBox.Pick('Pick box from screen', m);
```

Previous()

Description

Returns the previous box in the model.

Arguments

No arguments

Returns

MorphBox object (or null if there are no more boxs in the model).

Return type

MorphBox

Example

To get the box in model m before box b:

```
var b = b.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the boxs in the model.

Arguments

- **Model** (Model)

Model that all boxs will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the boxs in model m, from 1000000:

```
MorphBox.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged boxs in the model.

Arguments

- **Model** (Model)

Model that all the flagged boxs will be renumbered in

- **flag** (Flag)

Flag set on the boxs that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the boxs in model m flagged with f, from 1000000:

```
MorphBox.RenumberFlagged(m, f, 1000000);
```

Reset(redraw (optional)*[boolean]*)

Description

Resets the morph box to its initial position and updates the coordinates of all its nodes.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to reset several boxes and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using Model.UpdateGraphics().

Returns

No return value

Example

To reset box b:

```
b.Reset();
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select boxs using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting boxs

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only boxs from that model can be selected. If the argument is a Flag then only boxs that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any boxs can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of boxs selected or null if menu cancelled

Return type

Number

Example

To select boxs from model m, flagging those selected with flag f, giving the prompt 'Select boxs':

```
MorphBox.Select(f, 'Select boxs', m);
```

To select boxs, flagging those selected with flag f but limiting selection to boxs flagged with flag l, giving the prompt 'Select boxs':

```
MorphBox.Select(f, 'Select boxs', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the box.

Arguments

- **flag** (Flag)

Flag to set on the box

Returns

No return value

Example

To set flag f for box b:

```
b.SetFlag(f);
```

SetMorphConnections(status*[boolean]*) [static]

Description

Turns Morph Connections on/off.

Arguments

- **status** (boolean)

true turns Morph Connections on. false turns Morph Connections off.

Returns

No return value.

Example

To turn Morph Connections on.

```
MorphBox.SetMorphConnections(true);
```

SetPointID(xindex*[integer]*, yindex*[integer]*, zindex*[integer]*, id*[integer]*)

Description

Replaces the morph point ID on the array, whose size depends on the orders in X, Y and Z directions, with the given new ID.

Arguments

- **xindex** (integer)

Index of the point in X direction. Note that indices start at 0, so it should be 0 for the points with the smallest parameteric X coordinate and box.nx-1 for the points with the highest X.

- **yindex** (integer)

Index of the point in Y direction. Note that indices start at 0, so it should be 0 for the points with the smallest parameteric Y coordinate and box.ny-1 for the points with the highest Y.

- **zindex** (integer)

Index of the point in Z direction. Note that indices start at 0, so it should be 0 for the points with the smallest parameteric Z coordinate and box.nz-1 for the points with the highest Z.

- **id** (integer)

New MorphPoint id.

Returns

No return value

Example

To replace the 2nd point on the edge along the local X direction and at lowest local Y and highest local Z coordinate with point 101:

```
box.SetPointID(1, 0, box.nz-1, 101);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the box. The box will be sketched until you either call MorphBox.Unsketch(), MorphBox.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the box is sketched. If omitted redraw is true. If you want to sketch several boxs and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch box b:

```
b.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged boxs in the model. The boxs will be sketched until you either call MorphBox.Unsketch(), MorphBox.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged boxs will be sketched in

- **flag** (Flag)

Flag set on the boxs that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the boxs are sketched. If omitted redraw is true. If you want to sketch flagged boxs several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all boxs flagged with flag in model m:

```
MorphBox.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of boxs in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing boxs should be counted. If false or omitted referenced but undefined boxs will also be included in the total.

Returns

number of boxs

Return type

Number

Example

To get the total number of boxs in model m:

```
var total = MorphBox.Total(m);
```

Unblank()

Description

Unblanks the box

Arguments

No arguments

Returns

No return value

Example

To unblank box b:

```
b.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the boxs in the model.

Arguments

- **Model** (Model)

Model that all boxs will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the boxs in model m:

```
MorphBox.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged boxs in the model.

Arguments

- **Model** (Model)

Model that the flagged boxs will be unblanked in

- **flag** (Flag)

Flag set on the boxs that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the boxs in model m flagged with f:

```
MorphBox.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the boxs in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all boxs will be unset in

- **flag** (Flag)

Flag to unset on the boxs

Returns

No return value

Example

To unset the flag f on all the boxs in model m:

```
MorphBox.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the box.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the box is unsketched. If omitted redraw is true. If you want to unsketch several boxs and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch box b:

```
b.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all boxs.

Arguments

- **Model** (Model)

Model that all boxs will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the boxs are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all boxs in model m:

```
MorphBox.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged boxs in the model.

Arguments

- **Model** (Model)

Model that all boxs will be unsketched in

- **flag** (Flag)

Flag set on the boxs that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the boxs are unsketched. If omitted redraw is true. If you want to unsketch

several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all boxs flagged with flag in model m:

```
MorphBox.UnsketchAll(m, flag);
```

UpdateParametricCoordinates()

Description

Recalculates parametric X, Y, Z coordinates for each node in the *SET_NODE_COLUMN associated with the morph box. This needs to be called whenever morph points on the box or their coordinates have been changed manually and you wish to keep all nodes at their intrinsic global X, Y, Z coordinates. Provided Morph Connections is on (see MorphBox.SetMorphConnections()), this will also force PRIMER to recalculate the parametric coordinates for any connections in the morph box next time one of its morph points is moved.

Arguments

No arguments

Returns

No return value

Example

To recalculate all X, Y and Z coordinates for box b:

```
b.UpdateParametricCoordinates();
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

MorphBox object.

Return type

MorphBox

Example

To check if MorphBox property b.example is a parameter by using the MorphBox.GetParameter() method:

```
if (b.ViewParameters().GetParameter(b.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for box with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The box object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for box b:

```
b.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for box b:

```
b.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for box. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for box b:

```
b.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this box.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for box b:

```
var xrefs = b.Xrefs();
```

---

toString()

Description

Creates a string containing the morph box data in keyword format. Note that this contains the keyword header and the keyword cards. See also MorphBox.Keyword() and MorphBox.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for morph box b in keyword format

```
var s = b.toString();
```

MorphFlow class

The MorphFlow class gives you access to morph flows in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model *[Model]*, redraw (optional) *[boolean]*)
- BlankFlagged(Model *[Model]*, flag *[Flag]*, redraw (optional) *[boolean]*)
- Create(Model *[Model]*, modal (optional) *[boolean]*)
- First(Model *[Model]*)
- FlagAll(Model *[Model]*, flag *[Flag]*)
- ForEach(Model *[Model]*, func *[function]*, extra (optional) *[any]*)
- GetAll(Model *[Model]*, property (optional) *[string]*)
- GetFlagged(Model *[Model]*, flag *[Flag]*, property (optional) *[string]*)
- GetFromID(Model *[Model]*, number *[integer]*)
- GetFromName(Model *[Model]*, morph flow name *[string]*)
- Last(Model *[Model]*)
- Pick(prompt *[string]*, limit (optional) *[Model or Flag]*, modal (optional) *[boolean]*, button text (optional) *[string]*)
- Select(flag *[Flag]*, prompt *[string]*, limit (optional) *[Model or Flag]*, modal (optional) *[boolean]*)
- SketchFlagged(Model *[Model]*, flag *[Flag]*, redraw (optional) *[boolean]*)
- Total(Model *[Model]*, exists (optional) *[boolean]*)
- UnblankAll(Model *[Model]*, redraw (optional) *[boolean]*)
- UnblankFlagged(Model *[Model]*, flag *[Flag]*, redraw (optional) *[boolean]*)
- UnflagAll(Model *[Model]*, flag *[Flag]*)
- UnsketchAll(Model *[Model]*, redraw (optional) *[boolean]*)
- UnsketchFlagged(Model *[Model]*, flag *[Flag]*, redraw (optional) *[boolean]*)

Member functions

- AssociateComment(Comment *[Comment]*)
- Blank()
- Blanked()
- Browse(modal (optional) *[boolean]*)
- ClearFlag(flag *[Flag]*)
- Copy(range (optional) *[boolean]*)
- DetachComment(Comment *[Comment]*)
- Edit(modal (optional) *[boolean]*)
- Error(message *[string]*, options (optional) *[object]*)
- Error(message *[string]*, details (optional) *[string]*) [deprecated]
- Flagged(flag *[Flag]*)
- GetComments()
- GetParameter(prop *[string]*)
- GetRow(row *[integer]*)
- GetValue(index *[integer]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- RemoveRow(row *[integer]*)
- RemoveValue(index *[integer]*)
- SetFlag(flag *[Flag]*)
- SetRow(row *[integer]*, data *[Array of data]*)
- SetValue(index *[integer]*, value *[real]*)
- Sketch(redraw (optional) *[boolean]*)
- Unblank()
- Unsketch(redraw (optional) *[boolean]*)
- ViewParameters()

- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

MorphFlow properties

Name	Type	Description
exists (read only)	logical	true if flow exists, false if referred to but not defined.
include	integer	The Include file number that the flow is in.
max	real	Maximum value for type set to "CONTINUOUS" or "STEP" when written as design variable for LS-OPT.
min	real	Minimum value for type set to "CONTINUOUS" or "STEP" when written as design variable for LS-OPT.
model (read only)	integer	The Model number that the flow is in.
name	string	Name of the morph flow. If the flow is used for applying LS-OPT variables, this should match the variable name in the listing file written by LS-OPT.
npoints (read only)	integer	Number of morph points referenced by the flow.
nvals (read only)	integer	Number of values in the list when type is set to "DISCRETE".
step	real	Step size for type set to "STEP" when written as design variable for LS-OPT.
type	string	Range type for the morph flow. This should be "CONTINUOUS", "STEP" or "DISCRETE" and may be used for LS-OPT when writing design variable files from morph flows.

Detailed Description

The MorphFlow class allows you to create, modify and manipulate morph flows. See the documentation below for more details.

Constructor

new MorphFlow(Model*[Model]*, name*[string]*)

Description

Create a new MorphFlow object.

Arguments

- **Model** (Model)

Model that morph flow will be created in

- **name** (string)

MorphFlow name

Returns

MorphFlow object

Return type

MorphFlow

Example

To create a new (empty) morph flow in model m with name 'depth'

```
var f = new MorphFlow(m, "depth");
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a flow.

Arguments

- **Comment** (Comment)

Comment that will be attached to the flow

Returns

No return value

Example

To associate comment c to the flow flow:

```
flow.AssociateComment(c);
```

Blank()

Description

Blanks the flow

Arguments

No arguments

Returns

No return value

Example

To blank flow flow:

```
flow.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flows in the model.

Arguments

- **Model** (Model)

Model that all flows will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the flows in model m:

```
MorphFlow.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged flows in the model.

Arguments

- **Model** (Model)

Model that all the flagged flows will be blanked in

- **flag** (Flag)

Flag set on the flows that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the flows in model m flagged with f:

```
MorphFlow.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the flow is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if flow flow is blanked:

```
if (flow.Blanked() ) do_something...
```

Browse(modal (optional)*[boolean]*)

Description

Starts an edit panel in Browse mode.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Browse flow flow:

```
flow.Browse();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the flow.

Arguments

- **flag** (Flag)

Flag to clear on the flow

Returns

No return value

Example

To clear flag f for flow flow:

```
flow.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the flow. The target include of the copied flow can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

MorphFlow object

Return type

MorphFlow

Example

To copy flow flow into flow z:

```
var z = flow.Copy();
```

Create(Model*[Model]*, modal (optional)*[boolean]*) [static]

Description

Starts an interactive editing panel to create a flow

Arguments

- **Model** (Model)

Model that the flow will be created in.

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

MorphFlow object (or null if not made).

Return type

MorphFlow

Example

To start creating a flow in model m:

```
var flow = MorphFlow.Create(m);
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a flow.

Arguments

- **Comment** (Comment)

Comment that will be detached from the flow

Returns

No return value

Example

To detach comment c from the flow flow:

```
flow.DetachComment(c);
```

Edit(modal (optional)*[boolean]*)

Description

Starts an interactive editing panel.

Arguments

- **modal (optional)** (boolean)

If this window is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the window will be modal.

Returns

no return value

Example

To Edit flow flow:

```
flow.Edit();
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for flow with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The flow object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for flow flow:

```
flow.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for flow flow:

```
flow.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for flow. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for flow flow:

```
flow.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first flow in the model.

Arguments

- **Model** (Model)

Model to get first flow in

Returns

MorphFlow object (or null if there are no flows in the model).

Return type

MorphFlow

Example

To get the first flow in model m:

```
var flow = MorphFlow.First(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the flows in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all flows will be flagged in

- **flag** (Flag)

Flag to set on the flows

Returns

No return value

Example

To flag all of the flows with flag f in model m:

```
MorphFlow.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the flow is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the flow

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if flow flow has flag f set on it:

```
if (flow.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each flow in the model.
Note that ForEach has been designed to make looping over flows as fast as possible and so has some limitations. Firstly, a single temporary MorphFlow object is created and on each function call it is updated with the current flow data. This means that you should not try to store the MorphFlow object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new flows inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all flows are in

- **func** (function)

Function to call for each flow

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the flows in model m:

```
MorphFlow.ForEach(m, test);
function test(flow)
{
// flow is MorphFlow object
}
```

To call function test for all of the flows in model m with optional object:

```
var data = { x:0, y:0 };
MorphFlow.ForEach(m, test, data);
function test(flow, extra)
{
// flow is MorphFlow object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of MorphFlow objects or properties for all of the flows in a model in PRIMER. If the optional property argument is not given then an array of MorphFlow objects is returned. If the property argument is given, that property value for each flow is returned in the array instead of a MorphFlow object

Arguments

- **Model** (Model)

Model to get flows from

- **property (optional)** (string)

Name for property to get for all flows in the model

Returns

Array of MorphFlow objects or properties

Return type

Array

Example

To make an array of MorphFlow objects for all of the flows in model m:

```
var a = MorphFlow.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each flow in model m:

```
var a = MorphFlow.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a flow.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the flow flow:

```
var comm_array = flow.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of MorphFlow objects for all of the flagged flows in a model in PRIMER If the optional property argument is not given then an array of MorphFlow objects is returned. If the property argument is given, then that property value for each flow is returned in the array instead of a MorphFlow object

Arguments

- **Model** (Model)

Model to get flows from

- **flag** (Flag)

Flag set on the flows that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged flows in the model

Returns

Array of MorphFlow objects or properties

Return type

Array

Example

To make an array of MorphFlow objects for all of the flows in model m flagged with f:

```
var flow = MorphFlow.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the flows in model m flagged with f:

```
var a = MorphFlow.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the MorphFlow object for a flow ID.

Arguments

- **Model** (Model)

Model to find the flow in

- **number** (integer)

number of the flow you want the MorphFlow object for

Returns

MorphFlow object (or null if flow does not exist).

Return type

MorphFlow

Example

To get the MorphFlow object for flow 100 in model m

```
var flow = MorphFlow.GetFromID(m, 100);
```

GetFromName(Model*[Model]*, morph flow name*[string]*) [static]

Description

Returns the stored MorphFlow object for a morph flow name. WARNING: This assumes that there is at most one morph flow with a given name. Otherwise this function only returns the first occurrence.

Arguments

- **Model** (Model)

Model to find the morph flow in

- **morph flow name** (string)

name of the morph flow you want the MorphFlow object for

Returns

MorphFlow object (or null if morph flow does not exist).

Return type

MorphFlow

Example

To get the MorphFlow object for flow "depth" in model m

```
var f = MorphFlow.GetFromName(m, "depth");
```

GetParameter(prop*[string]*)

Description

Checks if a MorphFlow property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the MorphFlow.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

flow property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if MorphFlow property flow.example is a parameter:

```
Options.property_parameter_names = true;
if (flow.GetParameter(flow.example) ) do_something...
Options.property_parameter_names = false;
```

To check if MorphFlow property flow.example is a parameter by using the GetParameter method:

```
if (flow.ViewParameters().GetParameter(flow.example) ) do_something...
```

GetRow(row*[integer]*)

Description

Returns the data for a row in the morph flow.

Arguments

- **row** (integer)

The row you want the data for. **Note row indices start at 0**.

Returns

An array of numbers containing the morph point ID at index 0 and the vector components at indices 1, 2, 3.

Return type

Number

Example

To get the data for the 2nd row in morph flow f:

```
var data = f.GetRow(1);
var point_id = data[0];
var dx = data[1];
var dy = data[2];
var dz = data[3];
```

GetValue(index*[integer]*)

Description

Get the value at given index on the morph flow with type "DISCRETE".

Arguments

- **index** (integer)

The index where you are extracting the value. **Note row indices start at 0**.

Returns

real

Return type

Number

Example

To get the 2nd value for morph flow f with type "DISCRETE":

```
var value = f.GetValue(1);
```

To get the last value on the list of values on f:

```
var value = f.GetValue(f.nvals - 1);
```

Keyword()

Description

Returns the keyword for this morph flow (*MORPH_FLOW). **Note that a carriage return is not added**. See also MorphFlow.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for morph flow f:

```
var key = f.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the morph flow. **Note that a carriage return is not added**. See also MorphFlow.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for morph flow f:

```
var cards = f.KeywordCards();
```

Last(Model*[Model]*) [static]

Description

Returns the last flow in the model.

Arguments

- **Model** (Model)

Model to get last flow in

Returns

MorphFlow object (or null if there are no flows in the model).

Return type

MorphFlow

Example

To get the last flow in model m:

```
var flow = MorphFlow.Last(m);
```

Next()

Description

Returns the next flow in the model.

Arguments

No arguments

Returns

MorphFlow object (or null if there are no more flows in the model).

Return type

MorphFlow

Example

To get the flow in model m after flow flow:

```
var flow = flow.Next();
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a flow.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only flows from that model can be picked. If the argument is a Flag then only flows that are flagged with *limit* can be selected. If omitted, or null, any flows from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

MorphFlow object (or null if not picked)

Return type

MorphFlow

Example

To pick a flow from model m giving the prompt 'Pick flow from screen':

```
var flow = MorphFlow.Pick('Pick flow from screen', m);
```

Previous()

Description

Returns the previous flow in the model.

Arguments

No arguments

Returns

MorphFlow object (or null if there are no more flows in the model).

Return type

MorphFlow

Example

To get the flow in model m before flow flow:

```
var flow = flow.Previous();
```

RemoveRow(row*[integer]*)

Description

Removes the data (a morph point ID and its three vector components) for a row in *MORPH_FLOW.

Arguments

- **row** (integer)

The row you want to remove the data for. **Note that row indices start at 0**.

Returns

No return value.

Example

To remove the second row of data for morph flow f:

```
f.RemoveRow(1);
```

RemoveValue(index*[integer]*)

Description

Removes the value at given index in *MORPH_FLOW with type "DISCRETE".

Arguments

- **index** (integer)

The index where you are removing the value. **Note that indices start at 0**.

Returns

No return value.

Example

To remove the second value for morph flow f:

```
f.RemoveValue(1);
```

To remove the last value for f:

```
f.RemoveValue(f.nvals - 1);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select flows using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting flows

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only flows from that model can be selected. If the argument is a Flag then only flows that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any flows can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of flows selected or null if menu cancelled

Return type

Number

Example

To select flows from model m, flagging those selected with flag f, giving the prompt ’Select flows’:

```
MorphFlow.Select(f, ’Select flows’, m);
```

To select flows, flagging those selected with flag f but limiting selection to flows flagged with flag l, giving the prompt ’Select flows’:

```
MorphFlow.Select(f, ’Select flows’, l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the flow.

Arguments

- **flag** (Flag)

Flag to set on the flow

Returns

No return value

Example

To set flag f for flow flow:

```
flow.SetFlag(f);
```

SetRow(row*[integer]*, data*[Array of data]*)

Description

Sets the data for a row in *MORPH_FLOW.

Arguments

- **row** (integer)

The row you want to set the data for. **Note that row indices start at 0**.

- **data** (Array of data)

The data you want to set the row to. It should be of length 4 having the morph point ID at index 0, and the vector components at indices 1, 2, 3.

Returns

No return value.

Example

To set the second point of the morph flow f to be morph point 11 with unit vector in X-direction:

```
var array = [11, 1.0, 0.0, 0.0];
f.SetRow(1, array);
```

To append a new row of data (using the same array of values):

```
f.SetRow(f.npoints, array);
```

SetValue(index*[integer]*, value*[real]*)

Description

Sets the value at given index in a *MORPH_FLOW with type "DISCRETE".

Arguments

- **index** (integer)

The row you want to set the data for. **Note that row indices start at 0**.

- **value** (real)

The new value to insert into the list.

Returns

No return value.

Example

To set the second value morph flow f to 20.0:

```
f.SetValue(1, 20.0);
```

To append the value 20.0 to the end of the list:

```
f.SetValue(f.nvals, 20.0);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the flow. The flow will be sketched until you either call MorphFlow.Unsketch(), MorphFlow.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the flow is sketched. If omitted redraw is true. If you want to sketch several flows and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch flow flow:

```
flow.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged flows in the model. The flows will be sketched until you either call MorphFlow.Unsketch(), MorphFlow.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged flows will be sketched in

- **flag** (Flag)

Flag set on the flows that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the flows are sketched. If omitted redraw is true. If you want to sketch flagged flows several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all flows flagged with flag in model m:

```
MorphFlow.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of flows in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing flows should be counted. If false or omitted referenced but undefined flows will also be included in the total.

Returns

number of flows

Return type

Number

Example

To get the total number of flows in model m:

```
var total = MorphFlow.Total(m);
```

Unblank()

Description

Unblanks the flow

Arguments

No arguments

Returns

No return value

Example

To unblank flow flow:

```
flow.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flows in the model.

Arguments

- **Model** (Model)

Model that all flows will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the flows in model m:

```
MorphFlow.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged flows in the model.

Arguments

- **Model** (Model)

Model that the flagged flows will be unblanked in

- **flag** (Flag)

Flag set on the flows that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the flows in model m flagged with f:

```
MorphFlow.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the flows in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all flows will be unset in

- **flag** (Flag)

Flag to unset on the flows

Returns

No return value

Example

To unset the flag f on all the flows in model m:

```
MorphFlow.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the flow.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the flow is unsketched. If omitted redraw is true. If you want to unsketch several flows and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch flow flow:

```
flow.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flows.

Arguments

- **Model** (Model)

Model that all flows will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the flows are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all flows in model m:

```
MorphFlow.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged flows in the model.

Arguments

- **Model** (Model)

Model that all flows will be unsketched in

- **flag** (Flag)

Flag set on the flows that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the flows are unsketched. If omitted redraw is true. If you want to unsketch

several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all flows flagged with flag in model m:

```
MorphFlow.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

MorphFlow object.

Return type

MorphFlow

Example

To check if MorphFlow property flow.example is a parameter by using the MorphFlow.GetParameter() method:

```
if (flow.ViewParameters().GetParameter(flow.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for flow with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The flow object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...

autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for flow flow:

```
flow.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for flow flow:

```
flow.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for flow. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for flow flow:

```
flow.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this flow.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for flow flow:

```
var xrefs = flow.Xrefs();
```

---

toString()

Description

Creates a string containing the morph flow data in keyword format. Note that this contains the keyword header and the keyword cards. See also MorphFlow.Keyword() and MorphFlow.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for morph flow f in keyword format

```
var s = f.toString();
```

MorphPoint class

The MorphPoint class gives you access to morph points in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- First(Model*[Model]*)
- FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- FlagAll(Model*[Model]*, flag*[Flag]*)
- ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*)
- GetAll(Model*[Model]*, property (optional)*[string]*)
- GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(Model*[Model]*, number*[integer]*)
- Last(Model*[Model]*)
- LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- MoveFlagged(Model*[Model]*, flag*[Flag]*, dx*[real]*, dy*[real]*, dz*[real]*)
- NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*)
- Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*)
- RenumberAll(Model*[Model]*, start*[integer]*)
- RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*)
- Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*)
- SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- Total(Model*[Model]*, exists (optional)*[boolean]*)
- UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)
- UnflagAll(Model*[Model]*, flag*[Flag]*)
- UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*)
- UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*)

Member functions

- AssociateComment(Comment*[Comment]*)
- Blank()
- Blanked()
- ClearFlag(flag*[Flag]*)
- Copy(range (optional)*[boolean]*)
- DetachComment(Comment*[Comment]*)
- Error(message*[string]*, options (optional)*[object]*)
- Error(message*[string]*, details (optional)*[string]*) [deprecated]
- Flagged(flag*[Flag]*)
- GetComments()
- GetParameter(prop*[string]*)
- Keyword()
- KeywordCards()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Sketch(redraw (optional)*[boolean]*)
- Unblank()
- Unsketch(redraw (optional)*[boolean]*)
- ViewParameters()
- Warning(message*[string]*, options (optional)*[object]*)
- Warning(message*[string]*, details (optional)*[string]*) [deprecated]
- Xrefs()
- toString()

MorphPoint properties

Name	Type	Description
exists (read only)	logical	true if point exists, false if referred to but not defined.
include	integer	The Include file number that the point is in.
label	integer	MorphPoint number.
model (read only)	integer	The Model number that the point is in.
x	real	X coordinate
y	real	Y coordinate
z	real	Z coordinate

Detailed Description

The MorphPoint class allows you to create, modify and manipulate morph points. See the documentation below for more details.

Constructor

new MorphPoint(Model*[Model]*, label*[integer]*, x*[real]*, y*[real]*, z*[real]*)

Description

Create a new MorphPoint object.

Arguments

- **Model** (Model)

Model that morph point will be created in

- **label** (integer)

MorphPoint number

- **x** (real)

X coordinate

- **y** (real)

Y coordinate

- **z** (real)

Z coordinate

Returns

MorphPoint object

Return type

MorphPoint

Example

To create a new morph point in model m with label 100, at coordinates (20, 40, 10)

```
var n = new MorphPoint(m, 100, 20, 40, 10);
```

Details of functions

AssociateComment(Comment*[Comment]*)

Description

Associates a comment with a point.

Arguments

- **Comment** (Comment)

Comment that will be attached to the point

Returns

No return value

Example

To associate comment c to the point p:

```
p.AssociateComment(c);
```

Blank()

Description

Blanks the point

Arguments

No arguments

Returns

No return value

Example

To blank point p:

```
p.Blank();
```

BlankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the points in the model.

Arguments

- **Model** (Model)

Model that all points will be blanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the points in model m:

```
MorphPoint.BlankAll(m);
```

BlankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Blanks all of the flagged points in the model.

Arguments

- **Model** (Model)

Model that all the flagged points will be blanked in

- **flag** (Flag)

Flag set on the points that you want to blank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To blank all of the points in model m flagged with f:

```
MorphPoint.BlankFlagged(m, f);
```

Blanked()

Description

Checks if the point is blanked or not.

Arguments

No arguments

Returns

true if blanked, false if not.

Return type

Boolean

Example

To check if point p is blanked:

```
if (p.Blanked() ) do_something...
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on the point.

Arguments

- **flag** (Flag)

Flag to clear on the point

Returns

No return value

Example

To clear flag f for point p:

```
p.ClearFlag(f);
```

Copy(range (optional)*[boolean]*)

Description

Copies the point. The target include of the copied point can be set using Options.copy_target_include.

Arguments

- **range (optional)** (boolean)

If you want to keep the copied item in the range specified for the current include. Default value is false. To set current include, use Include.MakeCurrentLayer().

Returns

MorphPoint object

Return type

MorphPoint

Example

To copy point p into point z:

```
var z = p.Copy();
```

DetachComment(Comment*[Comment]*)

Description

Detaches a comment from a point.

Arguments

- **Comment** (Comment)

Comment that will be detached from the point

Returns

No return value

Example

To detach comment c from the point p:

```
p.DetachComment(c);
```

Error(message*[string]*, options (optional)*[object]*)

Description

Adds an error for point with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **options (optional)** (object)

Options for the error. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The point object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed error message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add an error message "My custom error" for point p:

```
p.Error("My custom error");
```

To add an error message "My custom error" with an autofix "My custom autofix", calling function my_autofix, for point p:

```
p.Error("My custom error", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Error(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds an error for point. For more details on checking see the Check class.

Arguments

- **message** (string)

The error message to give

- **details (optional)** (string)

An optional detailed error message

Returns

No return value

Example

To add an error message "My custom error" for point p:

```
p.Error("My custom error");
```

First(Model*[Model]*) [static]

Description

Returns the first point in the model.

Arguments

- **Model** (Model)

Model to get first point in

Returns

MorphPoint object (or null if there are no points in the model).

Return type

MorphPoint

Example

To get the first point in model m:

```
var p = MorphPoint.First(m);
```

FirstFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the first free point label in the model. Also see MorphPoint.LastFreeLabel(), MorphPoint.NextFreeLabel() and Model.FirstFreeItemLabel().

Arguments

- **Model** (Model)

Model to get first free point label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *First free in layer* in editing panels). If omitted the whole model will be used (Equivalent to *First free* in editing panels).

Returns

MorphPoint label.

Return type

Number

Example

To get the first free point label in model m:

```
var label = MorphPoint.FirstFreeLabel(m);
```

FlagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the points in the model with a defined flag.

Arguments

- **Model** (Model)

Model that all points will be flagged in

- **flag** (Flag)

Flag to set on the points

Returns

No return value

Example

To flag all of the points with flag f in model m:

```
MorphPoint.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the point is flagged or not.

Arguments

- **flag** (Flag)

Flag to test on the point

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if point p has flag f set on it:

```
if (p.Flagged(f) ) do_something...
```

ForEach(Model*[Model]*, func*[function]*, extra (optional)*[any]*) [static]

Description

Calls a function for each point in the model.
Note that ForEach has been designed to make looping over points as fast as possible and so has some limitations. Firstly, a single temporary MorphPoint object is created and on each function call it is updated with the current point data. This means that you should not try to store the MorphPoint object for later use (e.g. in an array) as it is temporary.
Secondly, you cannot create new points inside a ForEach loop.

Arguments

- **Model** (Model)

Model that all points are in

- **func** (function)

Function to call for each point

- **extra (optional)** (any)

An optional extra object/array/string etc that will appended to arguments when calling the function

Returns

No return value

Example

To call function test for all of the points in model m:

```
MorphPoint.ForEach(m, test);
function test(p)
{
// p is MorphPoint object
}
```

To call function test for all of the points in model m with optional object:

```
var data = { x:0, y:0 };
MorphPoint.ForEach(m, test, data);
function test(p, extra)
{
// p is MorphPoint object
// extra is data
}
```

GetAll(Model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of MorphPoint objects or properties for all of the points in a model in PRIMER. If the optional property argument is not given then an array of MorphPoint objects is returned. If the property argument is given, that property value for each point is returned in the array instead of a MorphPoint object

Arguments

- **Model** (Model)

Model to get points from

- **property (optional)** (string)

Name for property to get for all points in the model

Returns

Array of MorphPoint objects or properties

Return type

Array

Example

To make an array of MorphPoint objects for all of the points in model m:

```
var a = MorphPoint.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for each point in model m:

```
var a = MorphPoint.GetAll(m, 'foo');
```

GetComments()

Description

Extracts the comments associated to a point.

Arguments

No arguments

Returns

Array of Comment objects (or null if there are no comments associated to the node).

Return type

Array

Example

To get the array of comments associated to the point p:

```
var comm_array = p.GetComments();
```

GetFlagged(Model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Returns an array of MorphPoint objects for all of the flagged points in a model in PRIMER If the optional property argument is not given then an array of MorphPoint objects is returned. If the property argument is given, then that property value for each point is returned in the array instead of a MorphPoint object

Arguments

- **Model** (Model)

Model to get points from

- **flag** (Flag)

Flag set on the points that you want to retrieve

- **property (optional)** (string)

Name for property to get for all flagged points in the model

Returns

Array of MorphPoint objects or properties

Return type

Array

Example

To make an array of MorphPoint objects for all of the points in model m flagged with f:

```
var p = MorphPoint.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'x' for a node) for all of the points in model m flagged with f:

```
var a = MorphPoint.GetFlagged(m, f, 'foo');
```

GetFromID(Model*[Model]*, number*[integer]*) [static]

Description

Returns the MorphPoint object for a point ID.

Arguments

- **Model** (Model)

Model to find the point in

- **number** (integer)

number of the point you want the MorphPoint object for

Returns

MorphPoint object (or null if point does not exist).

Return type

MorphPoint

Example

To get the MorphPoint object for point 100 in model m

```
var p = MorphPoint.GetFromID(m, 100);
```

GetParameter(prop*[string]*)

Description

Checks if a MorphPoint property is a parameter or not. Note that object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. For this function to work the JavaScript interpreter must use the parameter name instead of the value. This can be done by setting the Options.property_parameter_names option to true before calling the function and then resetting it to false afterwards.. This behaviour can also temporarily be switched by using the MorphPoint.ViewParameters() method and 'method chaining' (see the examples below).

Arguments

- **prop** (string)

point property to get parameter for

Returns

Parameter object if property is a parameter, null if not.

Return type

Parameter

Example

To check if MorphPoint property p.example is a parameter:

```
Options.property_parameter_names = true;
if (p.GetParameter(p.example) ) do_something...
Options.property_parameter_names = false;
```

To check if MorphPoint property p.example is a parameter by using the GetParameter method:

```
if (p.ViewParameters().GetParameter(p.example) ) do_something...
```

Keyword()

Description

Returns the keyword for this morph point (*MORPH_POINT). **Note that a carriage return is not added**. See also MorphPoint.KeywordCards()

Arguments

No arguments

Returns

string containing the keyword.

Return type

String

Example

To get the keyword for morph point p:

```
var key = p.Keyword();
```

KeywordCards()

Description

Returns the keyword cards for the morph point. **Note that a carriage return is not added**. See also MorphPoint.Keyword()

Arguments

No arguments

Returns

string containing the cards.

Return type

String

Example

To get the cards for morph point p:

```
var cards = p.KeywordCards();
```

Last(Model[*Model*]) [static]

Description

Returns the last point in the model.

Arguments

- **Model** (Model)

Model to get last point in

Returns

MorphPoint object (or null if there are no points in the model).

Return type

MorphPoint

Example

To get the last point in model m:

```
var p = MorphPoint.Last(m);
```

LastFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the last free point label in the model. Also see MorphPoint.FirstFreeLabel(), MorphPoint.NextFreeLabel() and see Model.LastFreeItemLabel()

Arguments

- **Model** (Model)

Model to get last free point label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest free in layer* in editing panels). If omitted the whole model will be used.

Returns

MorphPoint label.

Return type

Number

Example

To get the last free point label in model m:

```
var label = MorphPoint.LastFreeLabel(m);
```

MoveFlagged(Model*[Model]*, flag*[Flag]*, dx*[real]*, dy*[real]*, dz*[real]*) [static]

Description

This function moves a selection of flagged morph points by a given vector and interpolates the movement of other morph points in the same way as this happens on the interactive morph panel. Note that the interpolation depends on the settings which can be switched on the interactive morph panel or by preferences. To apply the movement to the nodes in the box(es), you will need to call MorphBox.ApplyMorphing() at least for all relevant boxes or (if that is easier) for all morph boxes in the model.

Arguments

- **Model** (Model)

Model that the flagged morph points are in

- **flag** (Flag)

Flag set on the morph points explicitly selected to move

- **dx** (real)

X component of vector to be moved along

- **dy** (real)

Y component of vector to be moved along

- **dz** (real)

Z component of vector to be moved along

Returns

No return value

Example

To move all morph points in model m flagged with flag by 10 units in global Y direction while interpolating the other morph points as given by button settings or preferences:

```
MorphPoint.MoveFlagged(m, flag, 0.0, 10.0, 0.0);
```

Next()

Description

Returns the next point in the model.

Arguments

No arguments

Returns

MorphPoint object (or null if there are no more points in the model).

Return type

MorphPoint

Example

To get the point in model m after point p:

```
var p = p.Next();
```

NextFreeLabel(Model*[Model]*, layer (optional)*[Include number]*) [static]

Description

Returns the next free (highest+1) point label in the model. Also see MorphPoint.FirstFreeLabel(), MorphPoint.LastFreeLabel() and Model.NextFreeItemLabel()

Arguments

- **Model** (Model)

Model to get next free point label in

- **layer (optional)** (Include number)

Include file (0 for the main file) to search for labels in (Equivalent to *Highest+1 in layer* in editing panels). If omitted the whole model will be used (Equivalent to *Highest+1* in editing panels).

Returns

MorphPoint label.

Return type

Number

Example

To get the next free point label in model m:

```
var label = MorphPoint.NextFreeLabel(m);
```

Pick(prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*, button text (optional)*[string]*) [static]

Description

Allows the user to pick a point.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only points from that model can be picked. If the argument is a Flag then only points that are flagged with *limit* can be selected. If omitted, or null, any points from any model can be selected. from any model.

- **modal (optional)** (boolean)

If picking is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the pick will be modal.

- **button text (optional)** (string)

By default the window with the prompt will have a button labelled 'Cancel' which if pressed will cancel the pick and return null. If you want to change the text on the button use this argument. If omitted 'Cancel' will be used.

Returns

MorphPoint object (or null if not picked)

Return type

MorphPoint

Example

To pick a point from model m giving the prompt 'Pick point from screen':

```
var p = MorphPoint.Pick('Pick point from screen', m);
```

Previous()

Description

Returns the previous point in the model.

Arguments

No arguments

Returns

MorphPoint object (or null if there are no more points in the model).

Return type

MorphPoint

Example

To get the point in model m before point p:

```
var p = p.Previous();
```

RenumberAll(Model*[Model]*, start*[integer]*) [static]

Description

Renumbers all of the points in the model.

Arguments

- **Model** (Model)

Model that all points will be renumbered in

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the points in model m, from 1000000:

```
MorphPoint.RenumberAll(m, 1000000);
```

RenumberFlagged(Model*[Model]*, flag*[Flag]*, start*[integer]*) [static]

Description

Renumbers all of the flagged points in the model.

Arguments

- **Model** (Model)

Model that all the flagged points will be renumbered in

- **flag** (Flag)

Flag set on the points that you want to renumber

- **start** (integer)

Start point for renumbering

Returns

No return value

Example

To renumber all of the points in model m flagged with f, from 1000000:

```
MorphPoint.RenumberFlagged(m, f, 1000000);
```

Select(flag*[Flag]*, prompt*[string]*, limit (optional)*[Model or Flag]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select points using standard PRIMER object menus.

Arguments

- **flag** (Flag)

Flag to use when selecting points

- **prompt** (string)

Text to display as a prompt to the user

- **limit (optional)** (Model or Flag)

If the argument is a Model then only points from that model can be selected. If the argument is a Flag then only points that are flagged with *limit* can be selected (*limit* should be different to *flag*). If omitted, or null, any points can be selected. from any model.

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in PRIMER until this window is dismissed). If omitted the selection will be modal.

Returns

Number of points selected or null if menu cancelled

Return type

Number

Example

To select points from model m, flagging those selected with flag f, giving the prompt 'Select points':

```
MorphPoint.Select(f, 'Select points', m);
```

To select points, flagging those selected with flag f but limiting selection to points flagged with flag l, giving the prompt 'Select points':

```
MorphPoint.Select(f, 'Select points', l);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on the point.

Arguments

- **flag** (Flag)

Flag to set on the point

Returns

No return value

Example

To set flag f for point p:

```
p.SetFlag(f);
```

Sketch(redraw (optional)*[boolean]*)

Description

Sketches the point. The point will be sketched until you either call MorphPoint.Unsketch(), MorphPoint.UnsketchAll(), Model.UnsketchAll(), or delete the model

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the point is sketched. If omitted redraw is true. If you want to sketch several points and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch point p:

```
p.Sketch();
```

SketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Sketches all of the flagged points in the model. The points will be sketched until you either call MorphPoint.Unsketch(), MorphPoint.UnsketchFlagged(), Model.UnsketchAll(), or delete the model

Arguments

- **Model** (Model)

Model that all the flagged points will be sketched in

- **flag** (Flag)

Flag set on the points that you want to sketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the points are sketched. If omitted redraw is true. If you want to sketch flagged points several times and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To sketch all points flagged with flag in model m:

```
MorphPoint.SketchFlagged(m, flag);
```

Total(Model*[Model]*, exists (optional)*[boolean]*) [static]

Description

Returns the total number of points in the model.

Arguments

- **Model** (Model)

Model to get total for

- **exists (optional)** (boolean)

true if only existing points should be counted. If false or omitted referenced but undefined points will also be included in the total.

Returns

number of points

Return type

Number

Example

To get the total number of points in model m:

```
var total = MorphPoint.Total(m);
```

Unblank()

Description

Unblanks the point

Arguments

No arguments

Returns

No return value

Example

To unblank point p:

```
p.Unblank();
```

UnblankAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the points in the model.

Arguments

- **Model** (Model)

Model that all points will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the points in model m:

```
MorphPoint.UnblankAll(m);
```

UnblankFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unblanks all of the flagged points in the model.

Arguments

- **Model** (Model)

Model that the flagged points will be unblanked in

- **flag** (Flag)

Flag set on the points that you want to unblank

- **redraw (optional)** (boolean)

If model should be redrawn or not. If omitted redraw is false. If you want to do several (un)blanks and only redraw after the last one then use false for all redraws apart from the last one. Alternatively you can redraw using View.Redraw().

Returns

No return value

Example

To unblank all of the points in model m flagged with f:

```
MorphPoint.UnblankFlagged(m, f);
```

UnflagAll(Model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the points in the model.

Arguments

- **Model** (Model)

Model that the defined flag for all points will be unset in

- **flag** (Flag)

Flag to unset on the points

Returns

No return value

Example

To unset the flag f on all the points in model m:

```
MorphPoint.UnflagAll(m, f);
```

Unsketch(redraw (optional)*[boolean]*)

Description

Unsketches the point.

Arguments

- **redraw (optional)** (boolean)

If model should be redrawn or not after the point is unsketched. If omitted redraw is true. If you want to unsketch several points and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch point p:

```
p.Unsketch();
```

UnsketchAll(Model*[Model]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all points.

Arguments

- **Model** (Model)

Model that all points will be unblanked in

- **redraw (optional)** (boolean)

If model should be redrawn or not after the points are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all points in model m:

```
MorphPoint.UnsketchAll(m);
```

UnsketchFlagged(Model*[Model]*, flag*[Flag]*, redraw (optional)*[boolean]*) [static]

Description

Unsketches all flagged points in the model.

Arguments

- **Model** (Model)

Model that all points will be unsketched in

- **flag** (Flag)

Flag set on the points that you want to unsketch

- **redraw (optional)** (boolean)

If model should be redrawn or not after the points are unsketched. If omitted redraw is true. If you want to unsketch several things and only redraw after the last one then use false for redraw and call View.Redraw().

Returns

No return value

Example

To unsketch all points flagged with flag in model m:

```
MorphPoint.UnsketchAll(m, flag);
```

ViewParameters()

Description

Object properties that are parameters are normally returned as the integer or float parameter values as that is virtually always what the user would want. This function temporarily changes the behaviour so that if a property is a parameter the parameter name is returned instead. This can be used with 'method chaining' (see the example below) to make sure a property argument is correct.

Arguments

No arguments

Returns

MorphPoint object.

Return type

MorphPoint

Example

To check if MorphPoint property p.example is a parameter by using the MorphPoint.GetParameter() method:

```
if (p.ViewParameters().GetParameter(p.example) ) do_something...
```

Warning(message*[string]*, options (optional)*[object]*)

Description

Adds a warning for point with an optional autofix. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **options (optional)** (object)

Options for the warning. If omitted the default values below will be used. The properties available are:

Object has the following properties:

Name	Type	Description
autofixFunction (optional)	function	An optional function to call to apply an autofix. If omitted no autofix will be possible (default). The function has no arguments and should return a string which describes what the autofix has done. This will be displayed by PRIMER. The point object is accessible in the function using the 'this' keyword. As this function can do **anything**, please use with extreme care to ensure that the objects and data in PRIMER remain valid and do not get corrupted. For example, an autofix should **not** delete another object that has an autofix which still has to run, or delete the model that is being checked! Hoist with his own petard...
autofixMessage (optional)	string	A message to describe an autofix. If omitted no autofix will be possible (default)
details (optional)	string	A detailed warning message. If omitted no detailed message will be given (default)

Returns

No return value

Example

To add a warning message "My custom warning" for point p:

```
p.Warning("My custom warning");
```

To add a warning message "My custom warning" with an autofix "My custom autofix", calling function my_autofix, for point p:

```
p.Warning("My custom warning", { autofixMessage: "My custom autofix",
autofixFunction: my_autofix} );
```

Warning(message*[string]*, details (optional)*[string]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a warning for point. For more details on checking see the Check class.

Arguments

- **message** (string)

The warning message to give

- **details (optional)** (string)

An optional detailed warning message

Returns

No return value

Example

To add a warning message "My custom warning" for point p:

```
p.Warning("My custom warning");
```

Xrefs()

Description

Returns the cross references for this point.

Arguments

No arguments

Returns

Xrefs object.

Return type

Xrefs

Example

To get the cross references for point p:

```
var xrefs = p.Xrefs();
```

---

toString()

Description

Creates a string containing the morph point data in keyword format. Note that this contains the keyword header and the keyword cards. See also MorphPoint.Keyword() and MorphPoint.KeywordCards().

Arguments

No arguments

Returns

string

Return type

String

Example

To get data for morph point p in keyword format

```
var s = p.toString();
```

Options class

The Options class enables you to access several options in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- OpenFileOptions()

Options class properties

Name	Type	Description
auto_confirm	logical	If true then PRIMER will automatically confirm (i.e. press the OK button) on (most) message boxes that are mapped. If false (default) then the message boxes will be shown and wait for the user to press a button. This option may be useful to help automate an operation where PRIMER would normally show a message box and wait for the user to press a button.
browse_ missing_ include_file	logical	If true (default) then PRIMER will popup a "BROWSE MISSING INCLUDE FILE" panel while reading the model. If false then it will throw an 'missing include file error' and continue reading the model.
copy_target_ include	integer	This option sets the target include files for copied keywords. If it is set to Include.COPY_ CURRENT (default) then copied keywords will go into the current layer. If it is set to Include.COPY_SOURCE, copied keywords will go into the include of the original element. An existing include file number can also be used if the copied keywords should go into a certain include.
dyna_version	string	The Ansys LS-DYNA version used to write keyword files. Can be "971R5", "971R4", "971R3", "970v6763" etc (use Utils.GetLSDYNAVersions() to get a full list, or see the version popup in Model->Write '>>> Ansys LS-DYNA output options'). See also Model.Write and Include.Write
edit_keep_ on_top	logical	If true edit panels created from the Edit() or Create() methods will be kept on top of other windows. If false (default) then they can be lowered.
exception_ messages	logical	If true (default) error messages will be printed to the dialogue box/stdout when an exception occurs in the API. If false they will not be printed. This option may be useful if you are using try/catch to manage exceptions and you do not want any error messages to be printed.
keyout_ binary	logical	If true then the output file will be written out in binary. If false (default) then an ascii file will be written.
keyout_ compress_ format	constant	This option can be used to specify the mode of compression. Can be Model.INDIVIDUAL_GZIP, Model.INDIVIDUAL_ZIP or Model.PACKAGED_ZIP
keyout_ compress_ level	integer	Compression level for .gz and .zip files. Must be in the range 1 to 9 with 1 being the least compression (fastest speed) to 9 being the greatest compression (slowest speed)
keyout_ compress_ switch	constant	Switch to set the compression during keyout. Can be Model.COMPRESS_KEEP (default), Model.COMPRESS_OFF or Model.COMPRESS_ON

keyout_i10	logical	If true then i10 format will be used to write the file. If false (default) then the normal Ansys LS-DYNA format will be used.
keyout_large	logical	If true then large format will be used to write the file. If false (default) then the normal Ansys LS-DYNA format will be used. Note that large format is only available from version R7.1 and above.
keyout_method	constant	The method used to write include files. Can be Include.MASTER_ONLY, Include.MERGE, Include.SELECT, Include.NOT_WRITTEN, Include.SUBDIR (default) or Include.SAME_DIR
keyout_parameter_values	logical	This option can be used to specify how parameters are written. If true then the underlying values of any parameters will be written when they are used in data fields rather than '&name'. If false (default) then '&name' will be written.
keyout_path_type	constant	The method used to write include paths. Can be Include.ABSOLUTE (default) or Include.RELATIVE
keyout_separator	constant	The directory separator used when writing include files. Can be Include.NATIVE (default), Include.UNIX or Include.WINDOWS
merge_set_collect	logical	If true then when merging models PRIMER will merge *SET_COLLECT cards which have the same label. If false (default) then they will be renumbered. This is also used with Model.ImportInclude. The default for this can be set using the primer*merge_set_collect preference.
model_tabs_active	logical	If true (default) then PRIMER will show model tabs in the object selection menu. If false then PRIMER will hide model tabs in object selection menu.
node_replace_asrg	logical	If true nodes in *AIRBAG_SHELL_REFERENCE_GEOMETRY can be replaced by node merge/replace. If false they will not be considered.
pick_window_position	constant or Window	Position that the pick window will be shown on the screen. It can be any combination (bitwise OR) of Window.LEFT, Window.CENTRE, Window.RIGHT, Window.TOP, Window.MIDDLE and Window.BOTTOM or a Window object. If a window obect is used the pick window will be shown in the middle of that window. The default is Window.RIGHT|Window.TOP.
property_parameter_names	logical	If true object properties which are parameters will be returned as parameter names. If false object properties which are parameters will be returned as parameter values.
reset_cwd	logical	If true then the current working directory will not be changed after selecting a file. If false (default) then the current working directory will be changed after selecting a file. This option only applies to Windows machines.

Properties for connections

Name	Type	Description
connection_angle_tol	real	The angle tolerance used for spotwelds in the connections algorithm
connection_edge_dist	real	The edge distance used in the connections algorithm
connection_file	string	The connection file to read/write
connection_max_thickness	real	The maximum thickness used in the connections algorithm
connection_model	integer	The model number selected to make connections in
connection_part	integer	The part ID selected for connections
connection_write_flag (read only)	integer	Flag that will be set on selected connections when writing. This can be used in the user JavaScript to write connections to find which are selected.
solid_spotweld_diameter	real	The default diameter of solid spotwelds.

spotweld_element_type	integer	The default type of spotweld to make. can be: Conx.SPOTWELD_BEAM, Conx.SPOTWELD_HEXA1, Conx.SPOTWELD_HEXA4, Conx.SPOTWELD_HEXA8, Conx.SPOTWELD_HEXA12 or Conx.SPOTWELD_HEXA16

Properties for graphics

Name	Type	Description
airbag_colour	integer	Airbag symbol colour
background_colour	integer	Colour of the background
contacts_colour	integer	Contact surface colour
contour_text_pt_size	integer	Contour bar text size (in pts)
contour_text_size	integer	This property is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Contour bar text size [deprecated]
date_size	integer	This property is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Size of date (clock) display [deprecated]
edge_angle	real	Feature edge critical angle
edges_ign_pt	integer	Option for choosing how to draw free edges (can be set to TRUE or FALSE)
extra_nodes_colour	integer	Constrained extra nodes colour
feature_line	integer	Switch ON/OFF feature line (can be set to TRUE or FALSE)
for_mom_colour	integer	Nodal force/moment colour
graticule_text_size	integer	Graticule text size
label_colour	integer	Colour of the label
label_pt_size	integer	Label size (in pts)
label_size	integer	This property is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Label size [deprecated]
node_colour	integer	Nodes colour
nrb_colour	integer	Nodal rigid body colour
overlay_colour	integer	Colour of the overlay
overlay_edges	integer	Option for setting the overlay edges value (can be set to 0,1 or 2)
rigid_bodies_colour	integer	Constrained rigid body colour
rot_vels_colour	integer	Rotational velocity colour
sketch_colour	integer	Colour of the sketch
spotweldbeam_colour_from_panels	integer	Spotweld beam/solid colour
spr_colour_from_node_sets	integer	Constrained SPR/SPR2/SPR3 colour
text_colour	integer	Colour of the text
timehist_blks_colour	integer	Time history block colour

title_date_pt_size	integer	Size of title & date (clock) display (in pts)
tracer_partl_colour	integer	Tracer particle colour
trans_vels_colour	integer	Translational velocity colour
x_sections_colour	integer	Cross-section colour

Properties for mass properties calculation

Name	Type	Description
mass_properties_centre_x	real	X-coordinate of user defined centre.
mass_properties_centre_y	real	Y-coordinate of user defined centre.
mass_properties_centre_z	real	Z-coordinate of user defined centre.
mass_properties_coordinate_system_type	integer	Coordinate system selection: Model.GLOBAL_AXES, Model.LOCAL_AXES, Model.PRINCIPAL_AXES.
mass_properties_include_attached_mass_deformable_elems	logical	Option to include lumped mass attached to the nodes of deformable elements. Default is FALSE.
mass_properties_include_attached_mass_rigid_elems	logical	Option to include lumped mass attached to the nodes of rigid elements. Default is FALSE.
mass_properties_include_timestep_mass	logical	Option to switch on/off inclusion of timestep addded mass. Default is FALSE.
mass_properties_inertia_center	integer	Option to set the centre used in inertia properties calculation. By default Centre at CofG is used. Available options are: Model.CENTRE_AT_COFG, Model.USER_DEFINED_CENTRE.
mass_properties_local_axes	integer	CSYS ID when using local axes.
mass_properties_rigid_part_constrained_parts	logical	Option to switch on/off mass of *CONSTRAINED_RIGID_BODIES associated with a rigid part. Default is FALSE.
mass_properties_rigid_part_extra_nodes	logical	Option to switch on/off mass of *CONSTRAINED_EXTRA_NODES associated with a rigid part. Default is FALSE.

Properties for nastran

Name	Type	Description
convert_rbe2_cnrb	logical	Convert all RBE2s to *CONSTRAINED_NODAL_RIGID_BODY
merge_rbe_nodes	logical	Merge duplicate RBE dependent nodes
retain_mid_nodes	logical	Retain mid-side nodes for higher order elements

Properties for ssh

Name	Type	Description
ssh_buffer_size	integer	The size of the buffer used (in kiloBytes) when transferring data to/from the remote machine in the Ssh class. Depending on your network and the size of the files you are transferring, changing this value may make file transfers quicker. The default value is 64(kB) but any value in the range 1(kB) to 1024(kB) is allowed.

Properties for widgets

Name	Type	Description
max_widgets	integer	The maximum number of Widgets that can be made for one Window. The default value is 1000
max_window_lines	integer	The maximum number of lines that can be made for a Window.Error(), Window.Information(), Window.Message(), Window.Question() or Window.Warning() window. The default value is 25

Detailed Description

The Options class is used to get/set options that PRIMER uses for certain functions. The options are available as **class** properties. See the documentation for more details. An example: Options.mass_properties_include_attached_mass_deformable_elems=true

Details of functions

OpenFileOptions() [static]

Description

Brings up a window to edit the settings for writing out *INCLUDE files. E.g. Absolute or Relative file paths.

Arguments

No arguments

Returns

No return value.

Example

To bring up the window.

```
Options.OpenFileOptions();
```

PopupWindow class

The PopupWindow class allows you to create popup windows for a graphical user interface. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Member functions

- Hide()

PopupWindow properties

Name	Type	Description
persistent	boolean	If the popup window will remain mapped when a button is pressed in it. By default (false) when a button is pressed in the popup window the popup will be unmapped. If set to true then the popup will remain mapped until the user clicks out of the window or hides it by calling Hide()

Detailed Description

The PopupWindow class allows you to make popup windows (that you can place Widgets in) and link them to Widgets. The popup window is then displayed by right clicking on the Widget the popup is linked to. The following very simple example shows how to create a popup window and link it to a label Widget.

```
// Create popup window
var pw = new PopupWindow();
// Create some widgets in the popup window
var pl = new Widget(pw, Widget.LABEL,   1, 30,  1,  7, "Label");
var pb = new Widget(pw, Widget.BUTTON,  1, 30,  7, 13, "Button");
var pt = new Widget(pw, Widget.TEXTBOX, 1, 30, 20, 26, "Textbox");
// Create window with title "Popup example" from 0.8-1.0 in x and 0.5-0.6 in y
var w = new Window("Popup example", 0.8, 1.0, 0.5, 0.6);
// Create label widget
var l = new Widget(w, Widget.LABEL,   1, 50, 1,  7, "Right click for popup...");
// link popup window to widget
l.popupWindow = pw;
// Assign the onPopup callback method to the function 'do_popup'
// This is only required if you want to make any changes before the popup
appears
l.onPopup  = do_popup;
// Show the widget and start event loop
w.Show();
////////////////////////////////////////////////////////////////////////////
function do_popup()
{
    Message("Showing popup");
}
```

See the documentation below and the Widget class for more details.

Constructor

new PopupWindow()

Description

Create a new PopupWindow object.

Arguments

No arguments

Returns

PopupWindow object

Return type

PopupWindow

Example

To create a PopupWindow containing the buttons "Create" and "Edit" and link it to button b:

```
var pw = new PopupWindow();
var c = new Widget(pw, Widget.BUTTON, 1, 30, 1, 7,  "Create");
var e = new Widget(pw, Widget.BUTTON, 1, 30, 7, 13, "Edit");
b.popupWindow = pw;
```

Details of functions

Hide()

Description

Hides (unmaps) the popup window.

Arguments

No arguments

Returns

No return value

Example

To hide popup window w:

```
w.Hide();
```

Ssh class

The Ssh class allows you to connect to a remote computer using ssh, scp and sftp commands. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Member functions

- AuthenticateWithPassword(password*[string]*)
- AuthenticateWithPublicKey(passphrase (optional)*[string]*)
- Execute(data*[object]*)
- Get(remote*[string]*, local*[string]*)
- Put(remote*[string]*, local*[string]*)
- SftpGet(remote*[string]*, local*[string]*)
- SftpList(remote*[string]*)
- SftpMkdir(remote*[string]*, mode*[constant]*)
- SftpPut(remote*[string]*, local*[string]*)
- SftpRmdir(remote*[string]*)

Ssh constants

Constants for file bits

Name	Description
Ssh.SETGROUP_BIT	Set group bit
Ssh.SETUID_BIT	Set uid bit
Ssh.STICKY_BIT	sticky bit

Constants for file types

Name	Description
Ssh.DIRECTORY	Directory
Ssh.FILE	Regular file
Ssh.SOCKET	Socket
Ssh.SYMBOLIC_LINK	Symbolic link

Constants for permissions

Name	Description
Ssh.GROUP_EXECUTE	Group has execute permission
Ssh.GROUP_READ	Group has read permission
Ssh.GROUP_WRITE	Group has write permission
Ssh.OTHER_EXECUTE	Others have execute permission
Ssh.OTHER_READ	Others have read permission

Ssh.OTHER_WRITE	Others have write permission
Ssh.OWNER_EXECUTE	Owner has execute permission
Ssh.OWNER_READ	Owner has read permission
Ssh.OWNER_WRITE	Owner has write permission

Detailed Description

The Ssh class gives you simple functions to do secure connections to a remote computer using ssh. Oasys LS-DYNA Environment software is built with the OpenSSH library to support the ssh, scp and sftp protocols. The basic workflow is to create a connection using the Ssh constructor, authenticate the connection either by using a password and AuthenticateWithPassword or with a public key and AuthenticateWithPublicKey then the method Execute can be used to execute commands on the remote machine, the methods Get and Put can be used to copy files to and from the remote machine using scp, and the commands SftpGet, SftpList, SftpMkdir, SftpPut and SftpRmdir can be used to perform secure file transfer commands.
ssh uses a public and private key pair to do communication. The software uses RSA for the private and public keys and stores them in the files id_rsa and id_rsa.pub in the .oasys_ssh directory of your home directory (C:\Users\your.name\.oasys_ssh on Windows by default). A key length of 2048 bits is recommended. You keep your private key secure in your .oasys_ssh directory but the public key can be copied to the authorized_keys file on remote machines so that authentication can be done etc. The software also maintains fingerprints for the machines you connect to to ensure that you are connecting to the machine that you think you are. The first time you connect to a machine you are asked to confirm the remote machine is correct and the software stores the fingerprint for it in the known_hosts file in your .oasys_ssh directory. For second and subsequent connections the software checks the fingerprint of the remote machine against the one it has stored and will only connect if it matches.
When creating a new ssh connection to a remote machine and transferring files a small 'buffer' is required to transfer the data. The size of this buffer can be controlled using the Options.ssh_buffer_size property **before** the Ssh object is created.

Constructor

new Ssh(hostname*[string]*, username*[string]*)

Description

Create a new Ssh object for secure communication to a remote computer.

Arguments

- **hostname** (string)

The hostname of the machine that you want to connect to

- **username** (string)

The username on the machine that you want to connect to

Returns

Ssh object

Return type

Ssh

Example

To create a connection to machine "example" as user "username"

```
var s = new Ssh("example", "username");
```

Details of functions

AuthenticateWithPassword(password*[string]*)

Description

Authenticate the connection using password.

Arguments

- **password** (string)

The password for the username on the remote machine

Returns

no return value

Example

To prompt the user for a password and authenticate using it in SSH connection s:

```
var password = Window.GetPassword("Enter Password to connect", "Password");
s.AuthenticateWithPassword(password);
```

AuthenticateWithPublicKey(passphrase (optional)*[string]*)

Description

Authenticate the connection using your public key. Your public key from the file .oasys_ssh/id_rsa.pub must be in the file .oasys_ssh/authorized_keys on the remote machine.

Arguments

- **passphrase (optional)** (string)

The passphrase for authentication on the remote machine if required

Returns

no return value

Example

Authenticate using your public key in SSH connection s:

```
s.AuthenticateWithPublicKey();
```

Execute(data*[object]*)

Description

Execute a command in the ssh session and get the standard output and error streams.

Arguments

- **data** (object)

Execute data

Object has the following properties:

Name	Type	Description
arguments (optional)	Array of strings	The arguments to pass to the command

command	string	The command you want to run

Returns

Object with the following properties:

Name	Type	Description
status	integer	The exit code from the command
stderr	string	The standard error output from the command
stdout	string	The standard output from the command

Return type

object

Example

To run command "example.bat" with arguments "foo" and "bar" in SSH connection s:

```
var output = s.Execute( { command: 'example.bat', arguments: [ 'foo', 'bar' ]}
);
var text   = output.stdout;
var errors = output.stderr;
var ecode  = output.status;
```

Get(remote*[string]*, local*[string]*)

Description

Gets a file from the ssh connection using scp.

Arguments

- **remote** (string)

The path of the remote file to get

- **local** (string)

The path of the local file to write

Returns

no return value

Example

To get the remote file "/path/to/file.txt", creating local file "C:\path\to\file.txt" in SSH connection s:

```
s.Get("/path/to/file.txt", "C:\\path\\to\\file.txt");
```

Put(remote*[string]*, local*[string]*)

Description

Puts a file on the remote ssh connection using scp.

Arguments

- **remote** (string)

The path of the remote file to put

- **local** (string)

The path of the local file to read

Returns

no return value

Example

To put the local file "C:\path\to\file.txt" to remote file "/path/to/file.txt" in SSH connection s:

```
s.Put("/path/to/file.txt", "C:\\path\\to\\file.txt");
```

SftpGet(remote*[string]*, local*[string]*)

Description

Gets a file from the ssh connection using sftp.

Arguments

- **remote** (string)

The path of the remote file to get

- **local** (string)

The path of the local file to write

Returns

no return value

Example

To get the remote file "file.txt", creating local file "C:\path\to\file.txt" in SSH connection s using sftp:

```
s.SftpGet("file.txt", "C:\\path\\to\\file.txt");
```

SftpList(remote*[string]*)

Description

Gets a listing from the ssh connection using sftp.

Arguments

- **remote** (string)

The remote path to get the listing from

Returns

Array of objects. Each object contains the following information for a file/directory:

Name	Type	Description
atime	integer	Access time for the file (seconds since epoch)
gid	integer	The group ID
info	constant	Bitwise information for the file/directory. See the permissions, file types and file bits constants
mtime	integer	Modification time for the file (seconds since epoch)
name	string	The name of the file/directory
size	integer	The size of the file
uid	integer	The user ID

Return type

object

Example

To get listing from the the remote path "temp" in SSH connection s using sftp:

```
var listing = s.SftpList("temp");
for (l=0; l<listing.length; l++)
{
    Message(listing[l].name + ":" + listing[l].size;
}
```

SftpMkdir(remote*[string]*, mode*[constant]*)

Description

Creates a directory in the remote ssh connection using sftp.

Arguments

- **remote** (string)

The remote directory to create

- **mode** (constant)

The mode/permissions for the directory. See the permissions constants for details. Note that the user's file-creation mask (umask) value will also be taken into account when creating the directory.

Returns

true if successful, false if not

Return type

Boolean

Example

To create the remote path "temp" with, create, write and execute permissions for only the owner, in SSH connection s using sftp:

```
var success = s.SftpMkdir("temp", Ssh.OWNER_READ|Ssh.OWNER_WRITE|Ssh.OWNER_
EXECUTE);
```

SftpPut(remote*[string]*, local*[string]*)

Description

Puts a file on the remote ssh connection using sftp.

Arguments

- **remote** (string)

The path of the remote file to put

- **local** (string)

The path of the local file to read

Returns

no return value

Example

To put the local file "C:\path\to\file.txt" to remote file "file.txt" in SSH connection s:

```
s.SftpPut("file.txt", "C:\\path\\to\\file.txt");
```

SftpRmdir(remote*[string]*)

Description

Deletes a directory in the remote ssh connection using sftp. If this fails it is probably because the directory is not empty.

Arguments

- **remote** (string)

The remote directory to delete

Returns

true if successful, false if not

Return type

Boolean

Example

To delete the remote path "temp" in SSH connection s using sftp:

```
var success = s.SftpRmdir("temp");
```

Utils class

The Utils class contains various useful utility functions. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Ascii85Decode(encoded*[string]*)
- Ascii85Encode(data*[ArrayBuffer]*, length (optional)*[integer]*)
- Build()
- CallPromiseHandlers()
- CheckinLicense(feature*[string]*)
- CheckoutLicense(feature*[string]*)
- GarbageCollect()
- GetLSDYNAVersions()
- HTMLBrowser()
- HiResTimer()
- PdfReader()
- SHA256(filename*[string]*)
- SHA512(filename*[string]*)
- Terminate()
- TimerResolution()
- UUID()
- Version()

Detailed Description

The Utils class is used to provide various useful functions.

Details of functions

Ascii85Decode(encoded*[string]*) [static]

Description

Decodes an ASCII85 encoded string. See Utils.Ascii85Encode() for details on the method.

Arguments

- **encoded** (string)

An ASCII85 encoded string

Returns

ArrayBuffer object

Return type

ArrayBuffer

Example

To decode an ASCII85 encoded string:

```
var decoded = Utils.Ascii85Decode(encoded);
```

Ascii85Encode(data*[ArrayBuffer]*, length (optional)*[integer]*) [static]

Description

Encodes an ASCII85 encoded string. This enables binary data to be represented by ASCII characters using five ASCII characters to represent four bytes of binary data (making the encoded size 1/4 larger than the original). By doing this binary data can be stored in JavaScript strings. Note that the method used by PRIMER to encode and decode strings differs from the standard ASCII85 encoding as that uses the ASCII characters ", ’ and \ which cannot be used in JavaScript strings as they have special meanings. The method in PRIMER uses
0-84 are !-u (ASCII codes 33-117) (i.e. 33 is added to it) with the following exceptions
v is used instead of " (ASCII code 118 instead of 34)
w is used instead of ’ (ASCII code 119 instead of 39)
x is used instead of \ (ASCII code 120 instead of 92)
If all five digits are 0 they are represented by a single character z instead of !!!!!

Arguments

- **data** (ArrayBuffer)

ArrayBuffer containing the data

- **length (optional)** (integer)

Length of data in array buffer to encode. If omitted the whole array buffer will be encoded

Returns

string

Return type

String

Example

To encode ArrayBuffer data:

```
var encoded = Utils.Ascii85Encode(data);
```

Build() [static]

Description

Returns the build number

Arguments

No arguments

Returns

integer

Return type

Number

Example

To get the current build number

```
var build = Utils.Build();
```

CallPromiseHandlers() [static]

Description

Manually call any promise handlers/callbacks in the job queue

Arguments

No arguments

Returns

no return value

Example

To run any queued promise handlers/callbacks:

```
Utils.CallPromiseHandlers();
```

CheckinLicense(feature*[string]*) [static]

Description

Checks a license for a feature back in

Arguments

- **feature** (string)

feature to check license back in for

Returns

no return value

Example

To check in a license for "EXAMPLE":

```
Utils.CheckinLicense("EXAMPLE");
```

CheckoutLicense(feature*[string]*) [static]

Description

Checks out a license for a feature

Arguments

- **feature** (string)

feature to check license for

Returns

true if license available, false if not

Return type

Boolean

Example

To checkout a license for "EXAMPLE":

```
var got = Utils.CheckoutLicense("EXAMPLE");
if (got == false) Exit();
```

GarbageCollect() [static]

Description

Forces garbage collection to be done. This should not normally need to be called but in exceptional circumstances it can be called to ensure that garbage collection is done to return memory.

Arguments

No arguments

Returns

no return value

Example

To force garbage collection to be done:

```
Utils.GarbageCollect();
```

GetLSDYNAVersions() [static]

Description

Returns an array of all Ansys LS-DYNA output version names available in PRIMER.

Arguments

No arguments

Returns

An array of Ansys LS-DYNA version names

Return type

Array

Example

To get all Ansys LS-DYNA version names in PRIMER and set the output version to the latest version:

```
var versions = Utils.GetLSDYNAVersions();
      var latest = versions.length-1;

      Options.dyna_version = versions[latest];
```

HTMLBrowser() [static]

Description

Returns the path to the default HTML browser

Arguments

No arguments

Returns

string of the path

Return type

String

Example

To get path to the default HTML browser

```
var path = Utils.HTMLBrowser();
```

HiResTimer() [static]

Description

A high resolution timer that can be used to time how long things take. The first time this is called the timer will start and return 0. Subsequent calls will return the time in nanoseconds since the first call. Note that the timer will almost certainly not have 1 nanosecond precision but, depending on the platform, should should have a resolution of at least 1 microsecond. The resolution can be found by using Utils.TimerResolution()

Arguments

No arguments

Returns

number

Return type

number

Example

To time how long something takes to nanosecond precision:

```
var start = Utils.HiResTimer();
do something that takes some time...
var end = Utils.HiResTimer();
Message("it took " + (end-start) + "nanoseconds");
```

PdfReader() [static]

Description

Returns the path to the executable of the default pdf reader

Arguments

No arguments

Returns

string of the path

Return type

String

Example

To get path to the default pdf reader

```
var path = Utils.PdfReader();
```

SHA256(filename*[string]*) [static]

Description

Create a SHA-256 hash for a file

Arguments

- **filename** (string)

File to calculate the hash for

Returns

string

Return type

String

Example

To generate the hash for file "example.txt":

```
var sha256 = Utils.SHA256("example.txt");
```

SHA512(filename*[string]*) [static]

Description

Create a SHA-512 hash for a file

Arguments

- **filename** (string)

File to calculate the hash for

Returns

string

Return type

String

Example

To generate the hash for file "example.txt":

```
var sha512 = Utils.SHA512("example.txt");
```

Terminate() [static]

Description

Terminates the program. Note that this **terminates the entire program**, not the script. To terminate the script use Exit() instead

Arguments

No arguments

Returns

No return value

Example

To terminate the program:

```
Utils.Terminate();
```

TimerResolution() [static]

Description

Returns the resolution (precision) of the Utils.HiResTimer() timer in nanoseconds

Arguments

No arguments

Returns

number

Return type

number

Example

To find the resolution of the timer in nanoseconds:

```
var resolution = Utils.TimerResolution();
```

UUID() [static]

Description

Create an UUID (Universally Unique Identifier)

Arguments

No arguments

Returns

string

Return type

String

Example

To generate an UUID:

```
var uuid = Utils.UUID();
```

Version() [static]

Description

Returns the version number

Arguments

No arguments

Returns

real

Return type

Number

Example

To get the current version number

```
var version = Utils.Version();
```

View class

The View class allows you to control the view and plotting modes in PRIMER. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Ac()
- Ct()
- GetTargetEye()
- Hi()
- Li()
- Redraw()
- SetContourType(View type*[constant]*, View subtype*[constant]*, View subtype2*[constant]*)
- SetTargetEye(info*[object]*)
- Sh()
- Show(View type*[constant]*)
- Si()
- Vec()

View constants

Constants for SetContourType - argument 1 (Type)

Name	Description
View.ELEMPROPS	Type Element Properties
View.ELEMQUAL	Type Element Quality
View.INITVELS	Type Initial Velocities
View.LOADSHELLDIRECTION	Type Load Shell Direction
View.MASSSCALE	Type Mass Scale
View.MATLPROPS	Type Material Properties
View.PARTMASS	Type Part Mass
View.SHELLNORMALS	Type Shell Normals
View.SHELLTHICKNESS	Type Shell Thickness
View.TIMESTEP	Type TimeStep

Constants for SetContourType - argument 2 (Subtype)

Name	Description
View.ABSOLUTE	Subtype Absolute (of type Shell Thickness)
View.ADDEDMASS	Subtype Added Mass (of type Mass Scale)
View.ADDEDMASSPART	Subtype Added Mass #Part (of type Mass Scale)
View.AREA	Subtype Area (2d only) (of type Element Property)

View.ASPECTRATIO	Subtype Aspect Ratio (of type Element Quality)
View.CONTOUR	Subtype Contour (of type Shell Normals)
View.DENSITY	Subtype Density (of type Material Property)
View.EMPFINALMASS	Subtype (*)Final Mass (of type (*)EMP Parts Only)
View.EMPNSMASS	Subtype (*)NS Mass (of type (*)EMP Parts Only)
View.EMPSTRUCTMASS	Subtype (*)Struct Mass (of type (*)EMP Parts Only)
View.FAILEDCRITERIA	Subtype Failed Criteria (of type Element Quality)
View.FINALMASS	Subtype Final Mass (of type (*)EMP Parts Only)
View.FORM	Subtype Form (native) (of type Element Property)
View.FORMULATION	Subtype Formulation (of type Element Property)
View.INITVELRES	Subtype Init Vel-Res (of type Init Vel Component)
View.INITVELX	Subtype Init Vel-X (of type Init Vel Component)
View.INITVELY	Subtype Init Vel-Y (of type Init Vel Component)
View.INITVELZ	Subtype Init Vel-Z (of type Init Vel Component)
View.INTPOINTS	Subtype Integration Points (of type Element Property)
View.JACOBIAN	Subtype Jacobian (of type Element Quality)
View.MATERIALNUMBER	Subtype Material Number (of type Material Property)
View.MAXINTANGLE	Subtype Max Internal Angle (of type Element Quality)
View.MININTANGLE	Subtype Min Internal Angle (of type Element Quality)
View.MINLENGTH	Subtype Min Length (of type Element Quality)
View.PERCENTADDEDMASS	Subtype % Added Mass (of type Mass Scale)
View.PERCENTADDEDMASSPART	Subtype % Added Mass #Part (of type Mass Scale)
View.PLASTICSTRAIN	Subtype Plastic Strain (of type Element Property)
View.POISSONRATIO	Subtype Poisson’s Ratio (of type Material Property)
View.QUALIMPERF	Subtype Tet Collapse (of type Element Quality)
View.REMAINING	Subtype % remaining (of type Shell Thickness)
View.SKEW	Subtype Skew (native) (of type Element Quality)
View.STRUCTMASS	Subtype Struct Mass (of type (*)EMP Parts Only)
View.TAPER	Subtype Taper (of type Element Quality)
View.TETCOLLAPSE	Subtype Formulation (of type Element Quality)
View.THINNING	Subtype % thinning (of type Shell Thickness)
View.VECTOR	Subtype Vector (of type Shell Normals)
View.VOLUME	Subtype Volume (of type Element Property)
View.WARPAGE	Subtype Warpage (of type Element Quality)
View.YIELDSTRESS	Subtype Yield Stress (of type Material Property)
View.YOUNGMODULUS	Subtype Young’s Modulus (of type Material Property)

Constants for SetContourType - argument 3 (Subtype)

Name	**Description**

View.INTPOINT	Subtype Integration point (of type Element Property)
View.MAXSTRAIN	Subtype Maximum Strain (of type Element Property)
View.MINSTRAIN	Subtype Minimum Strain (of type Element Property)
View.PARAMETRICCOORD	Subtype Parametric coordinate (of type Element Property)

Constants for Show

Name	Description
View.ISO	Isometric projection
View.XY	XY axis projection
View.XZ	XZ axis projection
View.YZ	YZ axis projection

Constants for up vector

Name	Description
View.UP_AUTOMATIC	Automatically calculate up vector for target and eye
View.UP_X	Set up vector for target and eye to global X
View.UP_Y	Set up vector for target and eye to global Y
View.UP_Z	Set up vector for target and eye to global Z

Detailed Description

The View class gives you access to the different plotting modes and views. See the documentation below for more details.

Details of functions

Ac() [static]

Description

Autoscales the view

Arguments

No arguments

Returns

No return value

Example

To autoscale

```
View.Ac();
```

Ct() [static]

Description

Does a contour plot

Arguments

No arguments

Returns

No return value

Example

To do a contour plot

```
View.Ct();
```

GetTargetEye() [static]

Description

Get the current target and eye settings

Arguments

No arguments

Returns

Object with the following properties:

Name	Type	Description
eye	Array of numbers	Array containing the coordinates of the eye position
perspective	boolean	Whether the perspective switch is active
scale	real	Scale factor
target	Array of numbers	Array containing the coordinates of the target position
up	constant/array	View.UP_AUTOMATIC, View.UP_X, View.UP_Y, View.UP_Z or an array containing the up vector if user defined

Return type

object

Example

To get the current target and eye information

```
var info = View.GetTargetEye();
```

Hi() [static]

Description

Does a Hidden line plot

Arguments

No arguments

Returns

No return value

Example

To do a hidden line plot

```
View.Hi();
```

Li() [static]

Description

Does a line (wireframe) plot

Arguments

No arguments

Returns

No return value

Example

To do a line plot

```
View.Li();
```

Redraw() [static]

Description

Redraws the plot using the current plot mode.

Arguments

No arguments

Returns

No return value

Example

To redraw

```
View.Redraw();
```

SetContourType(View type*[constant]*, View subtype*[constant]*, View subtype2*[constant]*) [static]

Description

Sets a contour type (and subtype)

Arguments

- **View type** (constant)

The type of contour to plot. Can be:
View.ELEMPROPS, View.ELEMQUAL, View.INITVELS, View.LOADSHELLDIRECTION, View.MASSSCALE, View.MATLPROPS, View.PARTMASS, View.SHELLNORMALS, View.SHELLTHICKNESS, View.TIMESTEP

- **View subtype** (constant)

The subtype of contour to plot.
Note: This second argument is NOT required for types TIMESTEP and LOADSHELLDIRECTION.
Subtypes for Type TIMESTEP:

No subtypes
Subtypes for Type SHELLTHICKNESS:
View.ABSOLUTE, View.REMAINING, View.THINNING
Subtypes for SHELLNORMALS:
View.CONTOUR, View.VECTOR
Subtypes for Type LOADSHELLDIRECTION:
No subtypes
Subtypes for Type ELEMPROPS:
View.AREA, View.FORM, View.FORMULATION, View.INTPOINTS, View.PLASTICSTRAIN, View.VOLUME
Subtypes for Type ELEMQUAL:
View.ASPECTRATIO, View.FAILEDCRITERIA, View.JACOBIAN, View.MAXINTANGLE, View.MININTANGLE, View.MINLENGTH, View.QUALIMPERF, View.SKEW, View.TAPER, View.TETCOLLAPSE, View.WARPAGE
Subtypes for Type MASSSCALE:
View.ADDEDMASS, View.ADDEDMASSPART, View.PERCENTADDEDMASS, View.PERCENTADDEDMASSPART
Subtypes for Type MATLPROPS:
View.DENSITY, View.MATERIALNUMBER, View.POISSONRATIO, View.YIELDSTRESS, View.YOUNGMODULUS
Subtypes for Type INITVELS:
View.INITVELX, View.INITVELY, View.INITVELZ, View.INITVELRES
Subtypes for Type PARTMASS:
View.EMPFINALMASS, View.EMPNSMASS, View.EMPSTRUCTMASS, View.FINALMASS, View.STRUCTMASS

- **View subtype2** (constant)

The subtype of contour to plot.
Note: This third argument is required only for ELEMENTPROP ->PLASTICSTRAIN/FORM/AREA/VOLUME.
The default is PARAMETRIC COORDINATE.
Subtypes for Type ELEMENTPROP -> PLASTICSTRAIN/FORM/AREA/VOLUME:
View.INTEGRATIONPOINT, View.MAXSTRAIN, View.MINSTRAIN, View.PARAMETRICCOORD

Returns

No return value

Example

To set a contour plot of Load Shell Direction (no subtype):

```
View.SetContourType(View.LOADSHELLDIRECTION);
```

To set a contour plot of Element Formulation (type: Elem Props, subtype: Formulation):

```
View.SetContourType(View.ELEMPROPS, View.FORMULATION);
```

SetTargetEye(info*[object]*) [static]

Description

Set the current target and eye settings

Arguments

- **info** (object)

Object containing the target and eye properties

Object has the following properties:

Name	Type	Description
eye	Array of numbers	Array containing the coordinates of the eye position
perspective	boolean	Whether the perspective switch is active. If omitted the default is true
scale	real	Scale factor. If omitted, the scaling is automatically calculated (in the same way as if running PRIMER interactively)

target	Array of numbers	Array containing the coordinates of the target position
up	constant/array	View.UP_AUTOMATIC, View.UP_X, View.UP_Y, View.UP_Z or an array containing the up vector if user defined. If omitted the default is View.UP_AUTOMATIC

Returns

No return value

Example

To set the eye position to (-2365.1, -373.2, 1081.1) looking at target (-2075.9, -367.7, 905.2) with the up vector in global Z, automatically turning on perspective and calculating the scale factor:

```
View.SetTargetEye( { eye: [-2365.1, -373.2, 1081.1], target: [-2075.9, -367.7, 905.2], up: View.UP_Z } );
```

To set the eye position to (-2365.1, -373.2, 1081.1) looking at target (-2075.9, -367.7, 905.2) with the up vector in global Z, turning off perspective and setting the scale factor to 10:

```
View.SetTargetEye( { eye: [-2365.1, -373.2, 1081.1], target: [-2075.9, -367.7, 905.2], up: View.UP_Z, perspective: false, scale: 10 } );
```

Sh() [static]

Description

Does a shaded plot

Arguments

No arguments

Returns

No return value

Example

To do a shaded plot

```
View.Sh();
```

Show(View type*[constant]*) [static]

Description

Redraws using one of the standard views

Arguments

- **View type** (constant)

The view to show. Can be +/-View.XY, +/-View.YZ, +/-View.XZ or +/-View.ISO

Returns

No return value

Example

To do an isometric view from the negative direction:

```
View.Show(-View.ISO);
```

Si() [static]

Description

Does a shaded image contour plot

Arguments

No arguments

Returns

No return value

Example

To do a shaded image contour plot

```
View.Si();
```

Vec() [static]

Description

Does a vector plot

Arguments

No arguments

Returns

No return value

Example

To do a vector plot

```
View.Vec();
```

Widget class

The Widget class allows you to create components for a graphical user interface. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- CtrlPressed()
- PixelsPerUnit()
- ShiftPressed()
- StringLength(text*[string]*, monospace (optional)*[boolean]*, fontSize (optional)*[integer]*)

Member functions

- AddWidgetItem(item*[WidgetItem]*, position (optional)*[integer]*)
- AddWidgetItem(item*[WidgetItem]*, relationship*[constant]*, relitem*[WidgetItem]*)
- Circle(colour*[constant]*, fill*[boolean]*, xc*[integer]*, yc*[integer]*, radius*[integer]*)
- Clear()
- ClearSelection()
- Cross(colour (optional)*[constant]*)
- Delete()
- DirectoryIcon(line_colour*[constant]*, fill_colour*[constant]*)
- DumpImageString(filename*[string]*, format (optional)*[constant]*)
- Hide()
- ItemAt(index*[integer]*)
- Line(colour*[constant]*, x1*[integer]*, y1*[integer]*, x2*[integer]*, y2*[integer]*)
- MoveWidgetItem(item*[WidgetItem]*, relationship*[constant]*, relitem*[WidgetItem or null]*)
- Polygon(colour*[constant]*, fill*[boolean]*, points*[Array of numbers]*)
- Polygon(colour*[constant]*, fill*[boolean]*, x1*[integer]*, y1*[integer]*, x2*[integer]*, y2*[integer]*, ... xn*[integer]*, ... yn*[integer]*) [deprecated]
- ReadImageFile(filename*[string]*, justify (optional)*[constant]*, transparent (optional)*[colour value (integer)]*, tolerance (optional)*[integer]*)
- ReadImageString(string*[string]*, justify (optional)*[constant]*, transparent (optional)*[colour value (integer)]*, tolerance (optional)*[integer]*)
- Rectangle(colour*[constant]*, fill*[boolean]*, x1*[integer]*, y1*[integer]*, x2*[integer]*, y2*[integer]*)
- RemoveAllWidgetItems()
- RemoveWidgetItem(item*[WidgetItem]*)
- Scroll(scroll*[constant or WidgetItem object]*)
- Show()
- Static()
- Tick(colour (optional)*[constant]*)
- TotalItems()
- WidgetItems()

Widget constants

Name	Description
Widget.BUTTON	Button widget
Widget.CHECKBOX	Checkbox widget
Widget.COMBOBOX	Combobox widget
Widget.LABEL	Label widget
Widget.LISTBOX	Listbox widget

Widget.RADIOBUTTON	Radiobutton widget
Widget.SLIDER	Slider widget
Widget.TEXTBOX	Text input widget
Widget.TREE	Tree widget

Constants for Colour

Name	**Description**
Widget.BLACK	Colour black
Widget.BLUE	Colour blue
Widget.COLOUR_CONTRAST	A contrasting colour in the 3 user interface themes (Green, Purple, and Blue in the Dark, Light, and Classic themes respectively). Blue in the legacy theme.
Widget.COLOUR_CONTRAST_2	Another contrasting colour in the 3 user interface themes (Yellow, Red, and Red in the Dark, Light, and Classic themes respectively). Red in the legacy theme.
Widget.COLOUR_INVERSE	Inverse colour in the 3 user interface themes (Black or white depending on theme). Black in the legacy theme.
Widget.COLOUR_LABEL	Label text colour in the 3 user interface themes (Black or white depending on theme). Black in the legacy theme.
Widget.COLOUR_LATENT	Latent colour in the 3 user interface themes (Different shade of Cyan in every theme). Light Cyan in the legacy theme.
Widget.COLOUR_NEUTRAL	Neutral colour in the 3 user interface themes (Different shade of grey in every theme). Light grey in the legacy theme.
Widget.COLOUR_SAFE	Safe colour in the 3 user interface themes (Different shade of green in every theme). Dark green in the legacy theme.
Widget.COLOUR_TITLE	Title colour in the 3 user interface themes (Different shade of grey in every theme). Dark blue in the legacy theme.
Widget.COLOUR_WARNING	Warning colour in the 3 user interface themes (Different shade of red in every theme). Dark red in the legacy theme.
Widget.CYAN	Colour cyan
Widget.DARKBLUE	Colour dark blue
Widget.DARKGREEN	Colour dark green
Widget.DARKGREY	Colour dark grey
Widget.DARKGREY_NEUTRAL	Only valid in the function 'Line'. Used to keep the 3D effect in the legacy theme and not in the other themes. Neutral colour in the 3 user interface themes (Different shade of grey in every theme). Dark grey in the legacy theme
Widget.DARKRED	Colour dark red
Widget.DEFAULT	Default colour for widgets
Widget.GREEN	Colour green
Widget.GREY	Colour grey
Widget.LIGHTGREY	Colour light grey
Widget.LIGHTGREY_NEUTRAL	Only valid in the function 'Line'. Used to keep the 3D effect in the legacy theme and not in the other themes. Neutral colour in the 3 user interface themes (Different shade of grey in every theme). Light grey in the legacy theme
Widget.MAGENTA	Colour magenta
Widget.ORANGE	Colour orange

Widget.RED	Colour red
Widget.WHITE	Colour white
Widget.YELLOW	Colour yellow

Constants for Image RGB format

Name	Description
Widget.RGB24	24 bits for RGB data in widget images
Widget.RGB8	8 bits for RGB data in widget images

Constants for Justification

Name	Description
Widget.BOTTOM	Bottom justification
Widget.CENTRE	Centre (horizontal) justification
Widget.LEFT	Left justification
Widget.MIDDLE	Middle (vertical) justification
Widget.RIGHT	Right justification
Widget.SCALE	Image will be scaled to fit widget
Widget.TOP	Top justification

Constants for Orientation

Name	Description
Widget.HORIZONTAL	Horizontal orientation (for sliders)
Widget.VERTICAL	Vertical orientation (for sliders)

Constants for Selection

Name	Description
Widget.SELECT_ENHANCED	Multiple WidgetItems in a ListBox or tree Widget can be selected. When the user selects a WidgetItem the selection is cleared and the new WidgetItem selected. However, if the user presses the Ctrl key when clicking on a WidgetItem, the clicked WidgetItem gets toggled and all other WidgetItems are left untouched. If the user presses the Shift key while clicking on a WidgetItem, all WidgetItems between the last selected WidgetItem and the clicked WidgetItem are selected or unselected, depending on the state of the clicked WidgetItem.
Widget.SELECT_MULTIPLE	Multiple WidgetItems in a ListBox Widget can be selected. When the user selects a WidgetItem, the selection status of that WidgetItem is toggled and the other WidgetItems are left alone. Not valid for tree widgets. Widget.SELECT_ENHANCED will be used.
Widget.SELECT_NONE	No WidgetItem in a ListBox or tree Widget can be selected
Widget.SELECT_SINGLE	A single WidgetItem in a ListBox or tree Widget can be selected. When the user selects a WidgetItem, any already-selected WidgetItem becomes unselected, and the user cannot unselect the selected WidgetItem by clicking on it.

Constants for Tree relations

Name	Description

Widget.AFTER	Add a WidgetItem after the existing WidgetItem for a tree widget.
Widget.BEFORE	Add a WidgetItem before the existing WidgetItem for a tree widget.
Widget.CHILD	Add a WidgetItem as a child of the existing WidgetItem for a tree widget.

Constants for Tree/listbox scrolling

Name	Description
Widget.SCROLL_BOTTOM	Scroll tree or listbox widget to bottom.
Widget.SCROLL_DOWN	Scroll tree or listbox widget down one.
Widget.SCROLL_PAGE_DOWN	Scroll tree or listbox widget down one page.
Widget.SCROLL_PAGE_UP	Scroll tree or listbox widget up one page.
Widget.SCROLL_TOP	Scroll tree or listbox widget to top.
Widget.SCROLL_UP	Scroll tree or listbox widget up one.

Constants for User interface categories

Name	Description
Widget.CATEGORY_APPLY	Apply buttons
Widget.CATEGORY_BUTTON_BOX	A button box panel that contains other widgets
Widget.CATEGORY_CANCEL	Buttons which cancel the current operation
Widget.CATEGORY_DATA_ ENTRY_HEADER	Header for data entry cells, e.g. PRIMER create panels
Widget.CATEGORY_DISMISS	Buttons to close or dismiss panels
Widget.CATEGORY_ENTITY	Entity types in T/HIS
Widget.CATEGORY_GENERIC	A generic button that isn’t a special category
Widget.CATEGORY_GENERIC_2	An alternative to the generic category that has a complementary colour
Widget.CATEGORY_HELP	Help buttons
Widget.CATEGORY_KEYWORD	A PRIMER keyword button
Widget.CATEGORY_LABEL	A text label
Widget.CATEGORY_LABEL_BOX	Text label with a border
Widget.CATEGORY_LABEL_ POPUP	Text label with a popup that blends into the background
Widget.CATEGORY_MENU_BOX	A menu box
Widget.CATEGORY_MESSAGE	For displaying a temporary warning message
Widget.CATEGORY_OPERATE	Operate buttons in T/HIS
Widget.CATEGORY_POPUP_BOX	A popup box that can contain buttons and plain text
Widget.CATEGORY_SAFE_ ACTION	Buttons (usually green) to indicate a safe action
Widget.CATEGORY_SEL_ALL	Select all

Widget.CATEGORY_TAB	Tab
Widget.CATEGORY_TABLE_HEADER	Table (column) header
Widget.CATEGORY_TABLE_ROW	Table row
Widget.CATEGORY_TEXT_BOX	A text box
Widget.CATEGORY_TICKBOX	A tick box
Widget.CATEGORY_TITLE	Title text
Widget.CATEGORY_TOGGLE	Buttons that can be toggled, e.g. On/Off
Widget.CATEGORY_TOOL	Buttons within the tools area
Widget.CATEGORY_UNDO	Buttons which undo the last operation
Widget.CATEGORY_UNSEL_ALL	Unselect/deslect all
Widget.CATEGORY_UPDATE	Update buttons which update the screen but leave the panel open
Widget.CATEGORY_WARNING_ACTION	Buttons (usually red) to indicate a dangerous action
Widget.NO_CATEGORY	No styling is applied. Widget colour controlled by foreground/background properties and is the same in all themes

Widget properties

Name	Type	Description
active	logical	If widget is active (true) or disabled (false)
arrows	boolean	Whether arrows will be shown for a slider (default is true). Slider Widgets only.
background	constant	Widget background colour. Can be: Widget.BLACK, Widget.WHITE, Widget.RED, Widget.GREEN, Widget.BLUE, Widget.CYAN, Widget.MAGENTA, Widget.YELLOW, Widget.DARKRED, Widget.DARKGREEN, Widget.DARKBLUE, Widget.GREY, Widget.DARKGREY, Widget.LIGHTGREY, Widget.ORANGE, Widget.DEFAULT, Widget.COLOUR_NEUTRAL, Widget.COLOUR_CONTRAST, Widget.COLOUR_CONTRAST_2, Widget.COLOUR_WARNING, Widget.COLOUR_SAFE, Widget.COLOUR_TITLE, Widget.COLOUR_INVERSE, Widget.DARKGREY_NEUTRAL, Widget.LIGHTGREY_NEUTRAL Widget.COLOUR_LATENT, ,or a colour returned by Colour.RGB(). Note, background colours in the Window.THEME_DARK, Window.THEME_LIGHT, and Window.THEME_CLASSIC themes will be determined by the category of the widget not the background colour. To override this behaviour and use this background colour first set the widget category to Widget.NO_CATEGORY.
bottom	integer	Widget bottom coordinate
category	constant	The button category which determines the button's appearance when using the new user interface, see Window.Theme()
currentItem	WidgetItem object	The current WidgetItem for a tree Widget. The current WidgetItem in a tree is shown with a dashed border.
fontSize	integer	Widget font size in points. Currently only supports the following sizes: 6, 7, 8, 10, 12, 14, 18, 24. Can be used only with Widget.LABEL and Widget.BUTTON. Both LATIN1 and UTF-8 encoding is supported on Windows but Linux only supports LATIN1 encoding at the moment.

foreground	constant	Widget foreground colour. Can be: Widget.BLACK, Widget.WHITE, Widget.RED, Widget.GREEN, Widget.BLUE, Widget.CYAN, Widget.MAGENTA, Widget.YELLOW, Widget.DARKRED, Widget.DARKGREEN, Widget.DARKBLUE, Widget.GREY, Widget.DARKGREY, Widget.LIGHTGREY, Widget.ORANGE, Widget.DEFAULT, Widget.COLOUR_NEUTRAL, Widget.COLOUR_CONTRAST, Widget.COLOUR_CONTRAST_2, Widget.COLOUR_WARNING, Widget.COLOUR_SAFE, Widget.COLOUR_TITLE, Widget.COLOUR_LABEL, Widget.COLOUR_INVERSE, Widget.DARKGREY_NEUTRAL, Widget.LIGHTGREY_NEUTRAL Widget.COLOUR_LATENT, ,or a colour returned by Colour.RGB(). Note, foreground colours in the Window.THEME_DARK, Window.THEME_LIGHT, and Window.THEME_CLASSIC themes will be determined by the category of the widget not the foreground colour. To override this behaviour and use this foreground colour first set the widget category to Widget.NO_CATEGORY.
hover	string	Widget hover text
imageHeight (read only)	integer	Height of widget image (pixels)
imageWidth (read only)	integer	Width of widget image (pixels)
justify	constant	Widget justification. Can be: Widget.LEFT, Widget.RIGHT or Widget.CENTRE (default).
left	integer	Widget left coordinate
lineWidth	integer	Width of lines when drawing graphics (initially 1; values 1-100 allowed).
macroTag	string	Tag to use for this widget when recording a macro. If empty then the text property value will be used.
maximum	integer	The maximum value allowed for a slider (default is 100). Slider Widgets only.
minimum	integer	The minimum value allowed for a slider (default is 0). Slider Widgets only.
monospace	boolean	true if the widget uses a monospace font instead of a proportional width font (default). Label and button Widgets only.
onChange	function	Function to call when the text in a TEXTBOX widget, the selection in a COMBOBOX widget or the value of a SLIDER is changed. The Widget object is accessible in the function using the 'this' keyword (see the example below for more details of how to define the function and how to use the 'this' keyword). To unset the function set the property to null. **Note that this function is called when the user actually types something into the textbox, selects an item in the combobox or moves the slider, NOT when the Widget.text or Widget.value property changes**.
onClick	function	Function to call when a BUTTON, CHECKBOX, COMBOBOX, LABEL, RADIOBUTTON or TREE widget is clicked. The Widget object is accessible in the function using the 'this' keyword (see the example below for more details of how to define the function and how to use the 'this' keyword). To unset the function set the property to null. **Note that this function is called when the user actually clicks on the button, NOT when the Widget.pushed property changes**. For the COMBOBOX widget the function is called **before** the list of items is mapped.
onPopup	function	Function to call when a BUTTON, LABEL, TEXTBOX or TREE widget is right clicked to map a popup. The Widget object is accessible in the function using the 'this' keyword. The PopupWindow can then be found by using the popupWindow property of the Widget. The function is called **before** the popup is mapped so you can change the widgets in the popup as required.
onTimer	function	Function to call for a widget when timerDelay ms have elapsed after setting this. Additionally if timerRepeat is set this function will be called repetitively, every timerDelay ms. The Widget object is accessible in the function using the 'this' keyword. To unset the function set the property to null. **Note that as soon as this property is set the timer starts!**

orientation	constant	The orientation of a slider. Can be: Widget.VERTICAL or Widget.HORIZONTAL (default). Slider Widgets only.
popupDirection	constant	How PopupWindow will be mapped relative to this widget. Can be Widget.LEFT, Widget.RIGHT, Widget.TOP or Widget.BOTTOM (default). For tree widgets this will be ignored as the popup is always shown on the WidgetItem that is right clicked.
popupSymbol	logical	TRUE (default) if a symbol will be shown for a PopupWindow.
popupWindow	PopupWindow object	PopupWindow for this Widget. Only available for Button, Label and Textbox Widgets. To remove a PopupWindow from a Widget set to null.
pushed	logical	If widget is pushed (true) or not (false). This only affects Widget.BUTTON with the Widget.toggle property set, and Widget.CHECKBOX widgets.
right	integer	Widget right coordinate
select	constant	Selection method for ListBox and tree Widgets. Can be: Widget.SELECT_NONE, Widget.SELECT_SINGLE or Widget.SELECT_MULTIPLE or Widget.SELECT_ENHANCED (default).
selectedItem	WidgetItem object	WidgetItem that is currently selected for a ComboBox or Radiobutton, Widget. If null no WidgetItem is selected. For a ListBox Widget this property contains the last WidgetItem that was (de)selected. To get a list of all of the selected WidgetItems use WidgetItems() to return all of the WidgetItems and inspect the WidgetItem selected property.
shown (read only)	boolean	true if the widget is visible. To alter the visibility of a widget use the Show() and Hide() methods.
step	integer	The step value of a slider (default is 1). Slider Widgets only.
text	string	Widget text. For a ComboBox Widget this will be the text for the currently selected WidgetItem
textHidden	boolean	true if the widget text is hidden and replaced by asterisks. This may be used to create textboxes to type passwords in. TextBox Widgets only.
timerDelay	integer	Delay in ms before the function set for onTimer will be called. The initial value is 1000 (ms). Also see timerRepeat.
timerRepeat	logical	If the function set for onTimer will be called once (false) or repeatedly (true). The initial value is false. Also see timerDelay.
toggle	logical	If widget can be toggled (true) or not (false). This only affects Widget.BUTTON widgets.
top	integer	Widget top coordinate
type (read only)	integer	Type of the widget. The widget type could be Widget.BUTTON, Widget.CHECKBOX, Widget.COMBOBOX, Widget.LABEL, Widget.LISTBOX, Widget.RADIOBUTTON, Widget.SLIDER, Widget.TEXTBOX or Widget.TREE
value	integer	The current value of a slider (initially will be the minimum value). Slider Widgets only.
window (read only)	Window object	The Window that this widget is defined in
xResolution	integer	X resolution of button when drawing lines, circles, polygons and rectangles (initially 100). X coordinates on the Widget can be from 0 (on the left of the widget) to xResolution (on the right of the widget). Available for Widget.LABEL and Widget.BUTTON Widgets.
yResolution	integer	Y resolution of button when drawing lines, circles, polygons and rectangles (initially 100). Y coordinates on the Widget can be from 0 (on the top of the widget) to yResolution (on the bottom of the widget). Available for Widget.LABEL and Widget.BUTTON Widgets.

Detailed Description

The Widget class allows you to create Widgets (buttons, textboxes etc) in a Window for a graphical user interface. Callback functions can be declared for widgets to give actions when a button is pressed or the text in a textbox is selected etc. The following example displays various widgets in a window. Several callback methods are used. The exit button allows the user to exit the script but the button is only made active if the checkbox widget is ticked. If the button widgets are pressed feedback is given to the user

```
var count = 0;
// Create window
var w = new Window("Test", 0.8, 1.0, 0.5, 0.6);
// Create all of the widgets
var l = new Widget(w, Widget.LABEL,    1, 30, 1, 7, "Text:");
var t = new Widget(w, Widget.TEXTBOX, 31, 80, 1, 7, "Enter text");
var b = new Widget(w, Widget.BUTTON,   1, 30, 8, 14, "Press me");
var b2= new Widget(w, Widget.BUTTON,  31, 61, 8, 14, "Don't press me");
var c = new Widget(w, Widget.CHECKBOX,62, 68, 8, 14);
var l2= new Widget(w, Widget.LABEL,    1, 80, 15, 21, "You haven't pressed the
button yet...");
var e = new Widget(w, Widget.BUTTON,   1, 21, 22, 28, "Exit");
// Allow button widget b2 to toggle
b2.toggle = true;
// The exit button is initially inactive
e.active = false;
// Assign the callback functions
b.onClick  = clicked;
b2.onClick = clicked;
c.onClick  = clicked;
t.onChange = changed;
e.onClick  = confirm_exit;
// Show the window and start event loop
w.Show();
////////////////////////////////////////////////////////////////////////////
function clicked()
{
// If checkbox is clicked then set the state of the exit button
    if (this === c)
    {
        Message("Checkbox clicked");
        e.active = c.pushed;
    }
// If the "Don't press me' button is pressed then change the colour if the
button is pressed in.
    else if (this === b2)
    {
        Message("I said don't press!!!");
        if (b2.pushed) b2.background = Widget.WHITE;
        else           b2.background = Widget.DEFAULT;
    }
// If the "Press me" button is pressed then update the text in the label widget
// with how many times the button has been pressed.
    else
    {
        Message("You pressed...");
        count++;
        l2.text = "Button pressed " + count + " times";
    }
}
////////////////////////////////////////////////////////////////////////////
function changed()
{
// If the user has changed the text in the textbox then give a message in
// the dialogue box
    Message("Text has changed to " + this.text);
}
////////////////////////////////////////////////////////////////////////////
function confirm_exit()
{
// Map confirm box
    var ret = Window.Question("Confirm exit", "Are you sure you want to quit?");
```

```
// If the user has answered yes then exit from the script.
    if (ret == Window.YES) Exit();
}
```

In version 20 a tree widget was added. A simple tree widget example is shown below.

```
Window.Theme(Window.THEME_CURRENT);
let wi, cwi;
// Create a popup window and some widgets
let pw = new PopupWindow();
let pw_l1 = new Widget(pw, Widget.LABEL, 1, 61,  1,  7, "");
let pw_l2 = new Widget(pw, Widget.LABEL, 1, 61,  7, 13, "");
let pw_l3 = new Widget(pw, Widget.LABEL, 1, 61, 13, 19, "");
let pw_l4 = new Widget(pw, Widget.LABEL, 1, 61, 19, 25, "");
let pw_l5 = new Widget(pw, Widget.LABEL, 1, 61, 25, 31, "");
let pw_l6 = new Widget(pw, Widget.LABEL, 1, 61, 31, 37, "");
// Create window
let w = new Window("JavaScript Tree widget test", 0.85, 1.0, 0.75, 1.0);
// Create tree widget
let t = new Widget(w, Widget.TREE, 1, 61, 1, 51, "Suite");
// Add a root node to tree
let env_wi = new WidgetItem(t, "MyItem");
// Add a child to the root node
wi  = new WidgetItem(t, "PRIMER", Widget.CHILD, env_wi);
cwi = new WidgetItem(t, "Prepare", Widget.CHILD, wi);
wi.onMouseOver = wi_onmouseover;
cwi.hover = "Efficient, reliable model setup with support for all of the latest
Ansys LS-DYNA features";
wi  = new WidgetItem(t, "Ansys LS-DYNA", Widget.CHILD, env_wi);
cwi = new WidgetItem(t, "Analyse", Widget.CHILD, wi);
wi.onMouseOver = wi_onmouseover;
// Add a sibling node after Ansys LS-DYNA
wi  = new WidgetItem(t, "REPORTER", Widget.AFTER, wi);
cwi = new WidgetItem(t, "Report", Widget.CHILD, wi);
wi.onMouseOver = wi_onmouseover;
cwi.hover = "Automatic report generation for Ansys LS-DYNA simulations";

// Add a sibling node before REPORTER
wi  = new WidgetItem(t, "T/HIS", Widget.BEFORE, wi);
cwi = new WidgetItem(t, "Process", Widget.CHILD, wi);
wi.onMouseOver = wi_onmouseover;
cwi.hover = "Plot, manipulate and process XY data from Ansys LS-DYNA";

// Alternatively, create WidgetItem without parent and add
let d3plot_wi   = new WidgetItem(null, "D3PLOT");
t.AddWidgetItem(d3plot_wi, Widget.BEFORE, wi);
cwi = new WidgetItem(null, "Visualise");
t.AddWidgetItem(cwi, Widget.CHILD, d3plot_wi);
d3plot_wi.onMouseOver = wi_onmouseover;
cwi.hover = " In-depth 3D visualisation of Ansys LS-DYNA results";
// Expand root node
env_wi.expanded = true;
// Link popup to tree widget
t.popupWindow = pw;
// Assign callbacks
t.onClick = click_tree;
t.onPopup = do_popup;
// Show the widget and start event loop
w.Show();
////////////////////////////////////////////////////////////////////////////
function do_popup()
{
    pw_l1.text = this.currentItem.text;
    if (this.currentItem.selected) pw_l2.text = "Selected";
    else                           pw_l2.text = "Not selected";
    if (this.currentItem.Parent())
        pw_l3.text = "Parent: " + this.currentItem.Parent().text;
    else
        pw_l3.text = "No parent";
    if (this.currentItem.FirstChild())
        pw_l4.text = "First child: " + this.currentItem.FirstChild().text;
    else
```

```
        pw_l4.text = "No children";
    if (this.currentItem.PreviousSibling())
        pw_l5.text = "Previous: " + this.currentItem.PreviousSibling().text;
    else
        pw_l5.text = "No previous";
    if (this.currentItem.NextSibling())
        pw_l6.text = "Next: " + this.currentItem.NextSibling().text;
    else
        pw_l6.text = "No next";
}
////////////////////////////////////////////////////////////////////////////
function click_tree()
{
    Message("Clicked on "+this.currentItem.text+" in tree");
}
////////////////////////////////////////////////////////////////////////////
function wi_onmouseover()
{
    Message("Called onMouseOver for WidgetItem "+this.text+" in tree");
}
```

Graphics (lines, circles, rectangles etc) can be drawn on Widget.LABEL and Widget.BUTTON widgets. If these methods are used the resolution of the widget is 100 units in x and y and the origin is at the top left of the widget. See the documentation below and the WidgetItem and Window classes for more details.

Constructor

new Widget(window*[Window or PopupWindow]*, type*[constant]*, left*[integer]*, right*[integer]*, top*[integer]*, bottom*[integer]*, text (optional)*[string]*)

Description

Create a new Widget object.

Arguments

- **window** (Window or PopupWindow)

Window or PopupWindow that widget will be created in

- **type** (constant)

Widget type. Can be Widget.BUTTON, Widget.CHECKBOX, Widget.COMBOBOX, Widget.LABEL, Widget.LISTBOX, Widget.RADIOBUTTON, Widget.SLIDER, Widget.TEXTBOX or Widget.TREE

- **left** (integer)

left coordinate of widget

- **right** (integer)

right coordinate of widget

- **top** (integer)

top coordinate of widget

- **bottom** (integer)

bottom coordinate of widget

- **text (optional)** (string)

Text to show on widget (optional for LABEL, BUTTON, TEXTBOX and TREE, not required for CHECKBOX, COMBOBOX, LISTBOX, RADIOBUTTON and SLIDER). For a TREE widget the text will be used as a macroTag.

Returns

Widget object

Return type

Widget

Details of functions

AddWidgetItem(item*[WidgetItem]*, position (optional)*[integer]*)

Description

Adds a WidgetItem to a ComboBox ListBox or Radiobutton Widget. Also see Widget.RemoveAllWidgetItems and Widget.RemoveWidgetItem.

Arguments

- **item** (WidgetItem)

WidgetItem to add

- **position (optional)** (integer)

Position on Widget to add the WidgetItem. Any existing WidgetItems will be shifted down as required. If omitted the WidgetItem will be added to the end of the existing ones. **Note that positions start at 0**.

Returns

No return value

Example

To add WidgetItem wi to widget w:

```
w.AddWidgetItem(wi);
```

AddWidgetItem(item*[WidgetItem]*, relationship*[constant]*, relitem*[WidgetItem]*)

Description

Adds a WidgetItem to a Tree Widget. Also see Widget.RemoveAllWidgetItems and Widget.RemoveWidgetItem.

Arguments

- **item** (WidgetItem)

WidgetItem to add

- **relationship** (constant)

What relationship (relative to relitem) to use when adding item to the Widget. Can be:
Widget.BEFORE,
Widget.AFTER or
Widget.CHILD.

- **relitem** (WidgetItem)

Existing WidgetItem to add item relative to. If relationship is Widget.CHILD then relitem can be null and then the WidgetItem will be added to the root node of the tree.

Returns

No return value

Example

To add WidgetItem wi to tree widget w after existing WidgetItem ewi:

```
w.AddWidgetItem(wi, Widget.AFTER, ewi);
```

To add WidgetItem wi to tree widget w as a child of existing WidgetItem ewi:

```
w.AddWidgetItem(wi, Widget.CHILD, ewi);
```

Circle(colour*[constant]*, fill*[boolean]*, xc*[integer]*, yc*[integer]*, radius*[integer]*)

Description

Draws a circle on the widget. Only possible for Widget.LABEL and Widget.BUTTON widgets. The coordinates are local to the Widget, not the Window. See properties xResolution and yResolution for more details. Note that the widget graphics will only be updated when the widget is redrawn. This is to allow the user to do multiple drawing commands on a widget. To force the widget to be redrawn call Show().

Arguments

- **colour** (constant)

Colour of circle. See foreground for colours.

- **fill** (boolean)

If circle should be filled or not.

- **xc** (integer)

x coordinate of centre of circle.

- **yc** (integer)

y coordinate of centre of circle.

- **radius** (integer)

radius of circle.

Returns

no return value

Example

To draw a red filled circle, radius 25, at (50, 50) on widget w:

```
w.Circle(Widget.RED, true, 50, 50, 25);
```

Clear()

Description

Clears any graphics on the widget. Only possible for Widget.LABEL and Widget.BUTTON widgets. Note that the widget graphics will only be updated when the widget is redrawn. This is to allow the user to do multiple drawing commands on a widget. To force the widget to be redrawn call Show().

Arguments

No arguments

Returns

no return value

Example

To clear any graphics for widget w:

```
w.Clear();
```

ClearSelection()

Description

Clears selection of any WidgetItems on the widget. Only possible for Widget.COMBOBOX, Widget.LISTBOX, Widget.RADIOBUTTON and Widget.TREE widgets.

Arguments

No arguments

Returns

no return value

Example

To clear selection of any WidgetItems for widget w:

```
w.ClearSelection();
```

Cross(colour (optional)*[constant]*)

Description

Draws a cross symbol on the widget. Only possible for Widget.LABEL and Widget.BUTTON widgets.

Arguments

- **colour (optional)** (constant)

Colour of cross symbol. See foreground for colours. If omitted, current foreground colour is used.

Returns

no return value

Example

To draw a red cross symbol on widget w:

```
w.Cross(Widget.RED);
```

CtrlPressed() [static]

Description

Check to see if the Ctrl key is pressed

Arguments

No arguments

Returns

true/false

Return type

Boolean

Example

To test if someone has the Ctrl key pressed:

```
if (Widget.CtrlPressed()) { ... }
```

Delete()

Description

Deletes the widget from PRIMER (removing it from the window it is defined in) and returns any memory/resources used for the widget. This function should not normally need to be called. However, sometimes a script may want to recreate widgets in a window many times and unless the old widgets are deleted PRIMER will reach the maximum number of widgets for a window (Options.max_widgets). To avoid this problem this method can be used to force PRIMER to delete and return the resources for a widget. **Do not use the Widget object after calling this method**.

Arguments

No arguments

Returns

no return value

Example

To delete widget w:

```
w.Delete();
```

DirectoryIcon(line_colour*[constant]*, fill_colour*[constant]*)

Description

Draws a directory icon on the widget. Only possible for Widget.BUTTON widgets.

Arguments

- **line_colour** (constant)

Colour of lines of folder (only used in the old UI - in the new UI it will be ignored, a standard icon is always used). See foreground for colours.

- **fill_colour** (constant)

Colour of fill of folder (only used in the old UI - in the new UI it will be ignored, a standard icon is always used). See foreground for colours.

Returns

no return value

Example

To draw a directory icon on widget btn:

```
btn.DirectoryIcon(Widget.BLACK, Widget.YELLOW);
```

DumpImageString(filename*[string]*, format (optional)*[constant]*)

Description

Dumps a string representation of an image for a widget to a file in a form that can be used by Widget.ReadImageString(). Only possible for Widget.LABEL and Widget.BUTTON widgets.

Arguments

- **filename** (string)

Filename to dump string representation to

- **format (optional)** (constant)

Can be Widget.RGB8 or Widget.RGB24. Before version 15 PRIMER only used 8 bits to store RGB (red, green and

blue) colour information for widget images. In version 15 widget images have been changed to use 24 bits to store RGB information (8 bits for red, 8 bits for green and 8 bits for blue). Both formats are supported. If omitted the new Widget.RGB24 format will be used. See Widget.ReadImageString() for more details.

Returns

no return value

Example

To dump the image data to file 'image_data' for widget w with the old 8 bit RGB representation:

```
w.DumpImageString('image_data', Widget.RGB8);
```

To dump the image data to file 'image_data' for widget w with 24 bit RGB representation:

```
w.DumpImageString('image_data', Widget.RGB24);
```

Hide()

Description

Hides the widget on the screen

Arguments

No arguments

Returns

No return value

Example

To hide widget *w*

```
w.Hide();
```

ItemAt(index*[integer]*)

Description

Returns the WidgetItem object used at *index* in this Widget. See also Widget.TotalItems() and Widget.WidgetItems(). Note that for tree Widgets the items will not be returned in the order that they are displayed in, they will be returned in the order they were added to the tree.

Arguments

- **index** (integer)

index to return WidgetItem for. **Note that indices start at 0**.

Returns

WidgetItem object.

Return type

WidgetItem

Example

To loop over the WidgetItems used in Widget *w*

```
for (i=0; i<w.TotalItems(); i++)
{
    wi = w.ItemAt(i);
}
```

Line(colour*[constant]*, x1*[integer]*, y1*[integer]*, x2*[integer]*, y2*[integer]*)

Description

Draws a line on the widget. Only possible for Widget.LABEL and Widget.BUTTON widgets. The coordinates are local to the Widget, not the Window. See properties xResolution and yResolution for more details. Note that the widget graphics will only be updated when the widget is redrawn. This is to allow the user to do multiple drawing commands on a widget. To force the widget to be redrawn call Show().

Arguments

- **colour** (constant)

Colour of line. See foreground for colours.

- **x1** (integer)

x coordinate of start of line.

- **y1** (integer)

y coordinate of start of line.

- **x2** (integer)

x coordinate of end of line.

- **y2** (integer)

y coordinate of end of line.

Returns

no return value

Example

To draw a red line from (10, 90) to (90, 10) on widget w:

```
w.Line(Widget.RED, 10, 90, 90, 10);
```

MoveWidgetItem(item*[WidgetItem]*, relationship*[constant]*, relitem*[WidgetItem or null]*)

Description

Moves an existing WidgetItem in a tree Widget. Also see Widget.RemoveAllWidgetItems and Widget.RemoveWidgetItem.

Arguments

- **item** (WidgetItem)

WidgetItem to move

- **relationship** (constant)

What relationship (relative to relitem) to use when moving item to the Widget. Can be:
Widget.BEFORE,
Widget.AFTER or
Widget.AFTER.

- **relitem** (WidgetItem or null)

Existing WidgetItem to move item relative to. If relationship is Widget.CHILD then relitem can be null and then the WidgetItem will be moved to the root node of the tree.

Returns

No return value

Example

To move WidgetItem wi in tree widget w after existing WidgetItem ewi:

```
w.MoveWidgetItem(wi, Widget.AFTER, ewi);
```

To move WidgetItem wi in tree widget w as a child of existing WidgetItem ewi:

```
w.MoveWidgetItem(wi, Widget.CHILD, ewi);
```

PixelsPerUnit() [static]

Description

Returns the number of pixels per unit coordinate. This will vary depending on the monitor PRIMER is running on.

Arguments

No arguments

Returns

pixels/unit (real)

Return type

Number

Example

To return how many pixels there are per unit coordinate:

```
var ppu = Widget.PixelsPerUnit();
```

Polygon(colour*[constant]*, fill*[boolean]*, points*[Array of numbers]*)

Description

Draws a polygon on the widget. Only possible for Widget.LABEL and Widget.BUTTON widgets. The coordinates are local to the Widget, not the Window. See properties xResolution and yResolution for more details. Note that the widget graphics will only be updated when the widget is redrawn. This is to allow the user to do multiple drawing commands on a widget. To force the widget to be redrawn call Show(). The number of points (x, y pairs) is limited to 500. Any extra points will be ignored.

Arguments

- **colour** (constant)

Colour of polygon. See foreground for colours.

- **fill** (boolean)

If polygon should be filled or not.

- **points** (Array of numbers)

Array of point coordinates

Returns

no return value

Example

To draw a red filled triangle with corners (20, 20) and (50, 80) and (80, 20) on widget w:

```
var a = new Array(20, 20, 50, 80, 80, 20);
w.Polygon(Widget.RED, true, a);
```

Polygon(colour*[constant]*, fill*[boolean]*, x1*[integer]*, y1*[integer]*, x2*[integer]*, y2*[integer]*, ... xn*[integer]*, ... yn*[integer]*) [deprecated]

This function is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Draws a polygon on the widget. Only possible for Widget.LABEL and Widget.BUTTON widgets. The coordinates are local to the Widget, not the Window. See properties xResolution and yResolution for more details. Note that the widget graphics will only be updated when the widget is redrawn. This is to allow the user to do multiple drawing commands on a widget. To force the widget to be redrawn call Show(). The number of points (x, y pairs) is limited to 500. Any extra points will be ignored.

Arguments

- **colour** (constant)

Colour of polygon. See foreground for colours.

- **fill** (boolean)

If polygon should be filled or not.

- **x1** (integer)

x coordinate of point 1.

- **y1** (integer)

y coordinate of point 1.

- **x2** (integer)

x coordinate of point 2.

- **y2** (integer)

y coordinate of point 2.

- **... xn** (integer)

x coordinate of point n.

- **... yn** (integer)

y coordinate of point n.

Returns

no return value

Example

To draw a red filled triangle with corners (20, 20) and (50, 80) and (80, 20) on widget w:

```
w.Polygon(Widget.RED, true, 20, 20, 50, 80, 80, 20);
```

ReadImageFile(filename*[string]*, justify (optional)*[constant]*, transparent (optional)*[colour value (integer)]*, tolerance (optional)*[integer]*)

Description

Reads an image from a file to show on the widget. Only possible for Widget.LABEL and Widget.BUTTON widgets. The image will be shown on the widget underneath any text. Note that due to the way that colours are used for menus in PRIMER only a small number of colours are available for Widget images. Black and white images will display without any issues but colour images will be displayed with a reduced set of colours.

Arguments

- **filename** (string)

Image file (BMP, GIF, JPEG or PNG) to read. To remove an image use null.

- **justify (optional)** (constant)

Widget justification. Can be a bitwise or of Widget.LEFT, Widget.RIGHT or Widget.CENTRE and Widget.TOP, Widget.MIDDLE or Widget.BOTTOM. Additionally Widget.SCALE can be used to scale the image (either reducing or enlarging it) so that it fills the widget. If omitted the default is Widget.CENTRE|Widget.MIDDLE without scaling.

- **transparent (optional)** (colour value (integer))

Transparent colour. Must be a colour returned by Colour.RGB() in PRIMER. If given then this colour will be replaced by a transparent colour. i.e. the widget background colour will be shown. If omitted or null no transparency will be used.

- **tolerance (optional)** (integer)

Tolerance for transparent colour (0-255).
Any pixels in the image that have a red, green and blue colour value within *tolerance* of the transparent colour will be transparent.
For example if the transparent colour was given as Colour.RGB(255, 0, 0) and *tolerance* is 0 only pixels which have red value 255 **and** green value 0 **and** blue value 0 will be made transparent.
If *tolerance* is 4, pixels which have red values between 251 and 255 **and** green values between 0 and 4 **and** blue values between 0 and 4 will be made transparent.
If omitted a value of 8 will be used.

Returns

no return value

Example

To read image example.png for widget w and place it at the top left:

```
w.ReadImageFile("example.png", Widget.TOP|Widget.LEFT);
```

To read image example.png for widget w and place it at the top left, scaling it to fit the widget:

```
w.ReadImageFile("example.png", Widget.TOP|Widget.LEFT|Widget.SCALE);
```

To read image example.png for widget w and place it at the top left, replacing red with a transparent colour:

```
w.ReadImageFile("example.png", Widget.TOP|Widget.LEFT, Colour.RGB(255, 0, 0));
```

To remove an image from widget w:

```
w.ReadImageFile(null);
```

ReadImageString(string*[string]*, justify (optional)*[constant]*, transparent (optional)*[colour value (integer)]*, tolerance (optional)*[integer]*)

Description

Reads an image from a JavaScript string previously created by Widget.DumpImageString() to show on the widget. Only possible for Widget.LABEL and Widget.BUTTON widgets. The image will be shown on the widget underneath any text.
Note, prior to version 15 of PRIMER only a small number of colours were available for Widget images. In version 14 and earlier the RGB (red, green and blue) information for each pixel in the image was packed into a single byte (8 bits) with 3 bits for red, 3 for green and 2 for blue. Widget.DumpImageString() always returned the string beginning with "RRRGGGBB_RLE" which is this 8 bit format with run length encoding. This is format Widget.RGB8.
In version 15 support for Widget images was enhanced to give 24bit support for colours. The RGB information for each pixel has 8 bits for red, 8 bits for green and 8 bits for blue. This is format Widget.RGB24.
From version 15 Widget.DumpImageString() can either return the the old 8 bit format Widget.RGB8 (string beginning with "RRRGGGBB_RLE") or return the the new 24bit format Widget.RGB24 (string beginning with "RGB24_Z"). ReadImageString supports both formats.

Arguments

- **string** (string)

String containing the image data previously created by Widget.DumpImageString(). To remove an image use null.

- **justify (optional)** (constant)

Widget justification. Can be a bitwise or of Widget.LEFT, Widget.RIGHT or Widget.CENTRE and Widget.TOP, Widget.MIDDLE or Widget.BOTTOM. Additionally Widget.SCALE can be used to scale the image (either reducing or enlarging it) so that it fills the widget. If omitted the default is Widget.CENTRE|Widget.MIDDLE without scaling.

- **transparent (optional)** (colour value (integer))

Transparent colour. Must be a colour returned by Colour.RGB() in PRIMER. If given then this colour will be replaced by a transparent colour. i.e. the widget background colour will be shown. If omitted or null no transparency will be used.

- **tolerance (optional)** (integer)

Tolerance for transparent colour (0-255). Only used for the new 24bit format Widget.RGB24 (strings beginning with "RGB24_Z"). Ignored for the old 8 bit format Widget.RGB8 (strings beginning with "RRRGGGBB_RLE").
Any pixels in the image that have a red, green and blue colour value within *tolerance* of the transparent colour will be transparent.
For example if the transparent colour was given as Colour.RGB(255, 0, 0) and *tolerance* is 0 only pixels which have red value 255 **and** green value 0 **and** blue value 0 will be made transparent.
If *tolerance* is 4, pixels which have red values between 251 and 255 **and** green values between 0 and 4 **and** blue values between 0 and 4 will be made transparent.
If omitted a value of 8 will be used.

Returns

no return value

Example

To read image data from string s for widget w and place it at the top left:

```
w.ReadImageString(s, Widget.TOP|Widget.LEFT);
```

To read image data from string s for widget w and place it at the top left, scaling it to fit the widget:

```
w.ReadImageString(s, Widget.TOP|Widget.LEFT|Widget.SCALE);
```

To read image data from string s for widget w and place it at the top left, replacing red with a transparent colour:

```
w.ReadImageString(s, Widget.TOP|Widget.LEFT, Colour.RGB(255, 0, 0));
```

To remove an image from widget w:

```
w.ReadImageString(null);
```

Rectangle(colour*[constant]*, fill*[boolean]*, x1*[integer]*, y1*[integer]*, x2*[integer]*, y2*[integer]*)

Description

Draws a rectangle on the widget. Only possible for Widget.LABEL and Widget.BUTTON widgets. The coordinates are local to the Widget, not the Window. See properties xResolution and yResolution for more details. Note that the widget graphics will only be updated when the widget is redrawn. This is to allow the user to do multiple drawing commands on a widget. To force the widget to be redrawn call Show().

Arguments

- **colour** (constant)

Colour of rectangle. See foreground for colours.

- **fill** (boolean)

If rectangle should be filled or not.

- **x1** (integer)

x coordinate of first corner of rectangle.

- **y1** (integer)

y coordinate of first corner of rectangle.

- **x2** (integer)

x coordinate of second (opposite) corner of rectangle.

- **y2** (integer)

y coordinate of second (opposite) corner of rectangle.

Returns

no return value

Example

To draw a red filled rectangle with corners (20, 20) and (80, 80) on widget w:

```
w.Rectangle(Widget.RED, true, 20, 20, 80, 80);
```

RemoveAllWidgetItems()

Description

Removes any WidgetItems from the Widget. Also see Widget.AddWidgetItem and Widget.RemoveWidgetItem.

Arguments

No arguments

Returns

No return value

Example

To remove all WidgetItems from widget w:

```
w.RemoveAllWidgetItems();
```

RemoveWidgetItem(item*[WidgetItem]*)

Description

Removes a WidgetItem from the Widget. Also see Widget.AddWidgetItem and Widget.RemoveAllWidgetItems.

Arguments

- **item** (WidgetItem)

WidgetItem to remove

Returns

No return value

Example

To remove WidgetItem wi from widget w:

```
w.RemoveWidgetItem(wi);
```

Scroll(scroll*[constant or WidgetItem object]*)

Description

Scrolls a tree or listbox widget

Arguments

- **scroll** (constant or WidgetItem object)

How to scroll the tree/listbox widget. Can be: Widget.SCROLL_TOP, Widget.SCROLL_BOTTOM, Widget.SCROLL_UP, Widget.SCROLL_DOWN, Widget.SCROLL_PAGE_UP or Widget.SCROLL_PAGE_DOWN in which case the tree/listbox widget will be scrolled by that value or a WidgetItem, in which case the tree/listbox will be scrolled to make the WidgetItem visible, (expanding any branches as necessary to do so for the tree widget).

Returns

No return value

Example

To scroll tree widget w to the top:

```
w.Scroll(Widget.SCROLL_TOP);
```

To scroll listbox widget w so that WidgetItem wi is visible:

```
w.Scroll(wi);
```

ShiftPressed() [static]

Description

Check to see if the Shift key is pressed

Arguments

No arguments

Returns

true/false

Return type

Boolean

Example

To test if someone has the Shift key pressed:

```
if (Widget.ShiftPressed()) { ... }
```

Show()

Description

Shows the widget on the screen

Arguments

No arguments

Returns

No return value

Example

To show widget w:

```
w.Show();
```

Static()

Description

Windows have two different regions for Widgets. A 'normal' region which can be scrolled if required (if the window is made smaller scrollbars will be shown which can be used to scroll the contents) and a 'static' region at the top of the Window which is fixed and does not scroll. For an example of a static region in a Window see any of the keyword editing panels. The 'Dismiss', 'Create', 'Reset' etc buttons are in the static region. By default Widgets are put into the normal region of the Window. This method puts the Widget to the static region of the Window.

Arguments

No arguments

Returns

No return value

Example

To put widget w in the static part of the window:

```
w.Static();
```

StringLength(text*[string]*, monospace (optional)*[boolean]*, fontSize (optional)*[integer]*) [static]

Description

Returns the length of a string in Widget units. This can be used to find what size a Widget must be to be able to display the string.

Arguments

- **text** (string)

Text to find the width of

- **monospace (optional)** (boolean)

If true then width will be calculated using a monospace font. If false (default) then the normal proportional width font will be used

- **fontSize (optional)** (integer)

Calculation can be based on a defined font size, at the moment support is added only for font sizes of 6, 7, 8, 10, 12, 14, 18 and 24.

Returns

integer

Return type

Number

Example

To get the width of string 'Example':

```
var len = Widget.StringLength('Example');
```

Tick(colour (optional)*[constant]*)

Description

Draws a tick symbol on the widget. Only possible for Widget.LABEL and Widget.BUTTON widgets.

Arguments

- **colour (optional)** (constant)

Colour of tick symbol. See foreground for colours. If omitted, current foreground colour is used.

Returns

no return value

Example

To draw a red tick symbol on widget w:

```
w.Tick(Widget.RED);
```

TotalItems()

Description

Returns the number of the WidgetItem objects used in this Widget (or 0 if none used). See also Widget.ItemAt() and Widget.WidgetItems().

Arguments

No arguments

Returns

integer

Return type

Number

Example

To return the total number of WidgetItems used for Widget *w*

```
var total = w.TotalItems();
```

WidgetItems()

Description

Returns an array of the WidgetItem objects used in this Widget (or null if none used). See also Widget.ItemAt() and Widget.TotalItems().

Arguments

No arguments

Returns

Array of WidgetItem objects

Return type

Array

Example

To return WidgetItems used for Widget *w*

```
var wi = w.WidgetItems();
```

WidgetItem class

The WidgetItem class allows you to create items for combobox, listbox, radio button and tree Widgets. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Member functions

- FirstChild()
- NextSibling()
- Parent()
- PreviousSibling()

WidgetItem properties

Name	Type	Description
background	constant	Widget background colour. Not used for RADIOBUTTON widgets. Can be: Widget.BLACK, Widget.WHITE, Widget.RED, Widget.GREEN, Widget.BLUE, Widget.CYAN, Widget.MAGENTA, Widget.YELLOW, Widget.DARKRED, Widget.DARKGREEN, Widget.DARKBLUE, Widget.GREY, Widget.DARKGREY, Widget.LIGHTGREY or Widget.DEFAULT
expanded	logical	If the widget item is expanded (true) or not (false) in a tree. Only available for widgetitems used in TREE widgets.
foreground	constant	Widget foreground colour. Not used for RADIOBUTTON widgets. Can be: Widget.BLACK, Widget.WHITE, Widget.RED, Widget.GREEN, Widget.BLUE, Widget.CYAN, Widget.MAGENTA, Widget.YELLOW, Widget.DARKRED, Widget.DARKGREEN, Widget.DARKBLUE, Widget.GREY, Widget.DARKGREY, Widget.LIGHTGREY or Widget.DEFAULT
hover	string	WidgetItem's hover text. Not used for RADIOBUTTON widgets.
index (read only)	integer	The index of this widgetitem in the parent widget (undefined if widgetitem is not assigned to a widget).
monospace	boolean	true if the widgetitem uses a monospace font instead of a proportional width font (default). Not used for RADIOBUTTON and TREE widgets.
onClick	function	Function to call when a widget item in a COMBOBOX, LISTBOX or TREE widget is clicked. The Widgetitem object is accessible in the function using the 'this' keyword.
onMouseOver	function	Function to call when the mouse moves over a widget item in a COMBOBOX, LISTBOX or TREE widget. The Widgetitem object is accessible in the function using the 'this' keyword.
selectable	logical	If the widget item can be selected (true) or not (false). Not used for RADIOBUTTON and TREE widgets.
selected	logical	If the widget item is selected (true) or not (false).
text	string	Widget text. If the WidgetItem is used in a tree widget then the tree will not automatically redraw (this is in case you want to change lots of tree nodes at once). In this case, use the Window Redraw method to redraw the window.
visible	logical	If the widget item is visible (true) or not (false) in a tree. A widgetitem will not be visible if it is a child of a widgetitem that is not expanded. Only available for widgetitems used in TREE widgets.

widget (read only)	Widget object	The widget that this item is defined for (null if not set)

Detailed Description

The WidgetItem class allows you to create items for combobox, listbox, radio button and tree Widgets in a Window for a graphical user interface. The following example shows how WidgetItems are used to create a Combobox Widget and how to assign callbacks to determine when the selection has been changed.

```
var items = ["D3PLOT", "PRIMER", "SHELL", "REPORTER", "T/HIS"]
// Create window
var w = new Window("Combobox example", 0.8, 1.0, 0.5, 0.6);
// A simple combobox with a few items
var cl= new Widget(w, Widget.LABEL,   1, 30, 1, 7, "Programs:");
var cb= new Widget(w, Widget.COMBOBOX,  31, 61, 1, 7);
// Add WidgetItems to Combobox
for (i=0; i<items.length; i++)
    var wi = new WidgetItem(cb, items[i]);
// A combobox with many items showing a slider.
var li= new Widget(w, Widget.LABEL,   1, 30, 8, 14, "Long list:");
var ci= new Widget(w, Widget.COMBOBOX,  31, 61, 8, 14);
// Add WidgetItems to Combobox
// As an example we also make some of the WidgetItems unselectable and
// change the background colour
for (i=1; i<=100; i++)
{
    var wi = new WidgetItem(ci, "Item "+i);
    if ( (i % 10) == 5)
    {
        wi.selectable = false;
        wi.background = Widget.WHITE;
    }
}
var e = new Widget(w, Widget.BUTTON,   1, 21, 15, 21, "Exit");
// Assign callbacks
cb.onClick  = clicked;
cb.onChange = changed;
ci.onClick  = clicked;
ci.onChange = changed;
e.onClick   = confirm_exit
// Show the window and start event loop
w.Show();
////////////////////////////////////////////////////////////////////////////
function clicked()
{
// If combobox is clicked then print the current selection
    if (this.selectedItem)
        Message("selection is currently '"+this.selectedItem.text+"'");
}
////////////////////////////////////////////////////////////////////////////
function changed()
{
// If combobox selection is changed then print the new selection
    if (this.selectedItem)
        Message("selection is now '"+this.selectedItem.text+"'");
}
////////////////////////////////////////////////////////////////////////////
function confirm_exit()
{
// Map confirm box
    var ret = Window.Question("Confirm exit", "Are you sure you want to quit?");
// If the user has answered yes then exit from the script.
    if (ret == Window.YES) Exit();
}
```

See the documentation below and the Window and Widget classes for more details.

Constructor

new WidgetItem(widget*[Widget]*, text*[string]*, selectable (optional)*[boolean]*)

Description

Create a new WidgetItem object for a combobox, listbox or radio button widget.

Arguments

- **widget** (Widget)

Combobox, listbox or radio button Widget that the widget item will be created in. This can be null in which case the WidgetItem will be created but not assigned to a Widget. It can be assigned later by using Widget.AddWidgetItem().

- **text** (string)

Text to show on widget item

- **selectable (optional)** (boolean)

If the widget item can be selected. If omitted the widget item will be selectable. Not used for RADIOBUTTON and TREE widgets.

Returns

WidgetItem object

Return type

WidgetItem

new WidgetItem(widget*[Widget]*, text*[string]*, relationship (optional)*[constant]*, relitem (optional)*[WidgetItem]*)

Description

Create a new WidgetItem object for a tree widget. If the widget argument is given and the relationship and relitem arguments are omitted then the widget item will be added as a root node in the tree. If the relationship and relitem arguments are also used then the item can be added at a specific location in the tree. Alternatively, the widget argument can be null, and the relationship and relitem arguments omitted, in which case the WidgetItem will be created but not assigned to a Widget. It can be assigned to the tree later using Widget.AddWidgetItem()

Arguments

- **widget** (Widget)

Tree Widget that the widget item will be created in or null (if the relationship and relitem arguments are omitted)

- **text** (string)

Text to show on widget item

- **relationship (optional)** (constant)

What relationship (relative to relitem) to use when adding item to the Widget. Can be:
Widget.BEFORE,
Widget.AFTER or
Widget.CHILD.

- **relitem (optional)** (WidgetItem)

Existing WidgetItem to add item relative to

Returns

WidgetItem object

Return type

WidgetItem

Details of functions

FirstChild()

Description

Returns the first child WidgetItem or null if the WidgetItem has no children. Only possible for WidgetItems used in Widget.TREE widgets.

Arguments

No arguments

Returns

WidgetItem object

Return type

WidgetItem

Example

To get the first child widget item in a tree widget for widget item wi:

```
var cwi = wi.FirstChild();
```

NextSibling()

Description

Returns the next sibling WidgetItem or null if the WidgetItem has no further siblings. Only possible for WidgetItems used in Widget.TREE widgets.

Arguments

No arguments

Returns

WidgetItem object

Return type

WidgetItem

Example

To get the next sibling widget item in a tree widget for widget item wi:

```
var pwi = wi.NextSibling();
```

Parent()

Description

Returns the parent WidgetItem or null if the WidgetItem has no parent. Only possible for WidgetItems used in Widget.TREE widgets.

Arguments

No arguments

Returns

WidgetItem object

Return type

WidgetItem

Example

To get the parent widget item in a tree widget for widget item wi:

```
var pwi = wi.Parent();
```

PreviousSibling()

Description

Returns the next sibling WidgetItem or null if the WidgetItem has no further siblings. Only possible for WidgetItems used in Widget.TREE widgets.

Arguments

No arguments

Returns

WidgetItem object

Return type

WidgetItem

Example

To get the previous sibling widget item in a tree widget for widget item wi:

```
var pwi = wi.PreviousSibling();
```

Window class

The Window class allows you to create windows for a graphical user interface. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BottomBorder()
- BuildGUIFromString(guistring*[string]*)
- EndLoop()
- Error(title*[string]*, error*[string]*, buttons (optional)*[constant]*)
- GetDirectory(initial (optional)*[string]*)
- GetFile(extension (optional)*[string]*, save (optional)*[boolean]*, initial (optional)*[string]*)
- GetFilename(title*[string]*, message*[string]*, extension (optional)*[string]*, initial (optional)*[string]*, save (optional)*[boolean]*)
- GetFiles(extension (optional)*[string]*)
- GetGraphicsWindowPosition()
- GetInteger(title*[string]*, message*[string]*, initial (optional)*[integer]*)
- GetNumber(title*[string]*, message*[string]*, initial (optional)*[real]*)
- GetPassword(title*[string]*, message*[string]*)
- GetString(title*[string]*, message*[string]*, initial (optional)*[string]*)
- Information(title*[string]*, info*[string]*, buttons (optional)*[constant]*)
- MasterResolution()
- Message(title*[string]*, message*[string]*, buttons (optional)*[constant]*)
- MiddleBorder()
- Question(title*[string]*, question*[string]*, buttons (optional)*[constant]*)
- RightBorder()
- SetGraphicsWindowPosition(left*[real]*, right*[real]*, bottom*[real]*, top*[real]*)
- SetGraphicsWindowSize(width*[integer]*, height*[integer]*)
- StartLoop()
- Theme(theme (optional)*[constant]*)
- TopBorder()
- UpdateGUI()
- Warning(title*[string]*, warning*[string]*, buttons (optional)*[constant]*)

Member functions

- Delete()
- Hide()
- Recompute()
- Redraw()
- Show(modal (optional)*[boolean]*)

Window constants

Name	Description
Window.CANCEL	Show CANCEL button
Window.NO	Show NO button
Window.NONMODAL	Allow Window.Error, Window.Question, Window.Warning etc windows to be non modal
Window.OK	Show OK button
Window.YES	Show YES button

Constants for Resizing/positioning

Name	Description
Window.BOTTOM	Bottom resizing/positioning of window
Window.CENTRE	Centre (horizontal) positioning of window
Window.LEFT	Left resizing/positioning of window
Window.MIDDLE	Middle (vertical) positioning of window
Window.REDUCE	Window is allowed to reduce in size when resizing
Window.RIGHT	Right resizing/positioning of window
Window.TOP	Top resizing/positioning of window

Constants for Themes

Name	Description
Window.THEME_CLASSIC	Use the Classic theme (Note: Not only the script will use this theme, the whole interface of the program will switch to classic)
Window.THEME_CURRENT	Use the current theme
Window.THEME_DARK	Use the Dark theme (Note: Not only the script will use this theme, the whole interface of the program will switch to dark)
Window.THEME_LIGHT	Use the Light theme (Note: Not only the script will use this theme, the whole interface of the program will switch to light)
Window.USE_OLD_UI_JS	Use the original, pre v17, theme (default). (Note:The interface of the program will NOT switch to old)

Window properties

Name	Type	Description
active	boolean	If true (default) then the window then the window is active and widgets in the window can be used. If false then the window is inactive and the widgets cannot be used.
background	constant	Window background colour. Can be: Widget.BLACK, Widget.WHITE, Widget.RED, Widget.GREEN, Widget.BLUE, Widget.CYAN, Widget.MAGENTA, Widget.YELLOW, Widget.DARKRED, Widget.DARKGREEN, Widget.DARKBLUE, Widget.GREY, Widget.DARKGREY, Widget.LIGHTGREY or Widget.DEFAULT ,or a colour returned by Colour.RGB().
bottom	real	bottom coordinate of window in range 0.0 (bottom) to 1.0 (top)
height	real	height of window
keepOnTop	boolean	If true then the window will be kept "on top" of other windows. If false (default) then the window stacking order can be changed.
left	real	left coordinate of window in range 0.0 (left) to 1.0 (right)
maxWidgets (read only)	integer	The maximum number of widgets that can be made in this window. This can be changed **before** the window is created by using Options.max_widgets. Also see totalWidgets
onAfterShow	function	Function to call **after** a Window is shown. The Window object is accessible in the function using the 'this' keyword. This may be useful to ensure that certain actions are done after the window is shown. It can also be used to show another window so this enables multiple windows to be shown. To unset the function set the property to null.

onBeforeShow	function	Function to call **before** a Window is shown. The Window object is accessible in the function using the 'this' keyword. This may be useful to ensure that buttons are shown/hidden etc before the window is shown. Note that it cannot be used to show another window. Use onAfterShow for that. To unset the function set the property to null.
onClose	function	Function to call when a Window is closed by pressing the X on the top right of the window. This may be useful to ensure that certain actions are done when the window is closed. The Window object is accessible in the function using the 'this' keyword. If this function returns false then closing the window will be cancelled. Returning any other value (or not returning anything) will be ignored and the window will close. To unset the function set the property to null.
onHide	function	Function to call when a Window is hidden by calling Hide(). This may be useful to ensure that certain actions are done after the window is hidden. The Window object is accessible in the function using the 'this' keyword. To unset the function set the property to null.
resize	constant	Window resizing. By default when a Window is shown it is allowed to resize on all sides (left, right, top and bottom) to try to make enough room to show the Widgets. The behaviour can be changed by using this property. It can be any combination (bitwise OR) of Window.LEFT, Window.RIGHT, Window.TOP or Window.BOTTOM or 0. In addition Window.REDUCE can also be added to allow the window to reduce in size when resizing. Note that when Window.Show is called this property is set to 0 (i.e. not to resize on any side).
right	real	right coordinate of window in range 0.0 (left) to 1.0 (right)
showClose	boolean	If true (default) then a close (X) button will automatically be added on the top right of the window. If false then no close button will be shown.
shown (read only)	boolean	true if window is currently shown, false if not
title	string	Window title
top	real	top coordinate of window in range 0.0 (bottom) to 1.0 (top)
totalWidgets (read only)	integer	The total number of widgets that have been made in this window. This can be changed **before** the window is created by using Options.max_widgets. Also see maxWidgets
width	real	width of window

Detailed Description

The Window class allows you to make windows that you can place Widgets in to create a graphical user interface. The Widget class also gives a number of static methods for convenience. e.g. Window.GetInteger(). The following very simple example displays some text in a window with a button that unmaps the window when it is pressed and the user confirms that they want to exit.

```
// Create window with title "Text" from 0.8-1.0 in x and 0.5-0.6 in y
var w = new Window("Text", 0.8, 1.0, 0.5, 0.6);
// Create label widget
var l = new Widget(w, Widget.LABEL,   1, 40, 1,  7, "Press OK to exit");
// Create button widget
var e = new Widget(w, Widget.BUTTON, 11, 30, 8, 14, "OK");
// Assign the onClick callback method to the function confirm_exit'
e.onClick  = confirm_exit;
// Show the widget and start event loop
w.Show();
////////////////////////////////////////////////////////////////////////
function confirm_exit()
{
// Map confirm window
    var ret = Window.Question("Confirm exit", "Are you sure you want to quit?");
// If the user has answered Yes then exit.
    if (ret == Window.YES) Exit();
}
```

See the documentation below and the Widget class for more details.

Constructor

new Window(title*[string]*, left*[real]*, right*[real]*, bottom*[real]*, top*[real]*)

Description

Create a new Window object.

Arguments

- **title** (string)

Window title to show in title bar

- **left** (real)

left coordinate of window in range 0.0 (left) to 1.0 (right)

- **right** (real)

right coordinate of window in range 0.0 (left) to 1.0 (right)

- **bottom** (real)

bottom coordinate of window in range 0.0 (bottom) to 1.0 (top)

- **top** (real)

top coordinate of window in range 0.0 (bottom) to 1.0 (top)

Returns

Window object

Return type

Window

Example

To create a Window 'Example' in the top right half of the screen:

```
var w = new Window('Example', 0.5, 1.0, 0.5, 1.0);
```

Details of functions

BottomBorder() [static]

Description

Returns the vertical position of the bottom border (in range 0-1). This can be used to help position windows on the screen.

Arguments

No arguments

Returns

real in range 0-1

Return type

Number

Example

To obtain the position of the bottom border:

```
var b = Window.BottomBorder();
```

BuildGUIFromString(guistring*[string]*) [static]

Description

Builds a GUI from a JSON string created by the GUI builder.

Arguments

- **guistring** (string)

The string to create the GUI from

Returns

object containing the GUI

Return type

Object

Example

To create a GUI from the string json_string:

```
var gui = Window.BuildGUIFromString(json_string);
```

Delete()

Description

Deletes the window from PRIMER and returns any memory/resources used for the window. **This function should not normally need to be called**. However, in exceptional circumstances if a script recreates windows many times PRIMER may run out of USER objects on Microsoft Windows because of the way PRIMER creates and shows windows. To avoid this problem this method can be used to force PRIMER to return the resources for a window. **Do not use the Window object after calling this method**.

Arguments

No arguments

Returns

No return value

Example

To delete window w:

```
w.Delete();
```

EndLoop() [static]

Description

Ends an event loop started by Window.StartLoop()

Arguments

No arguments

Returns

No return value

Example

To end a blocking event loop started by Window.StartLoop():

```
Window.EndLoop();
```

Error(title*[string]*, error*[string]*, buttons (optional)*[constant]*) [static]

Description

Show an error message in a window.

Arguments

- **title** (string)

Title for window.

- **error** (string)

Error message to show in window. The maximum number of lines that can be shown is controlled by the Options.max_window_lines option.

- **buttons (optional)** (constant)

The buttons to use. Can be bitwise OR of Window.OK, Window.CANCEL, Window.YES or Window.NO. If this is omitted an OK button will be used. By default the window will be modal. If Window.NONMODAL is also given the window will be non-modal instead.

Returns

Button pressed

Return type

Number

Example

To show error *Critical error!\nAbort?* in window with title *Error* with Yes and No buttons:

```
var answer = Window.Error("Error", "Critical error!\nAbort?", Window.YES |
Window.NO);
if (answer == Window.YES) Exit();
```

GetDirectory(initial (optional)*[string]*) [static]

Description

Map the directory selector box native to your machine, allowing you to choose a directory. On Unix this will be a Motif selector. Windows will use the standard windows directory selector.

Arguments

- **initial (optional)** (string)

Initial directory to start from.

Returns

directory (string), (or null if cancel pressed).

Return type

String

Example

To select a directory:

```
var dir = Window.GetDirectory();
```

GetFile(extension (optional)*[string]*, save (optional)*[boolean]*, initial (optional)*[string]*) [static]

Description

Map a file selector box allowing you to choose a file. See also Window.GetFiles() and Window.GetFilename().

Arguments

- **extension (optional)** (string)

Extension to filter by.

- **save (optional)** (boolean)

If true the file selector is to be used for saving a file. If false (default) the file selector is for opening a file. Due to native operating system file selector differences, on linux new filenames can only be given when saving a file. On windows it is possible to give new filenames when opening or saving a file.

- **initial (optional)** (string)

Initial directory to start from.

Returns

filename (string), (or null if cancel pressed).

Return type

String

Example

To select a file using extension '.key':

```
var file = Window.GetFile(".key");
```

GetFilename(title*[string]*, message*[string]*, extension (optional)*[string]*, initial (optional)*[string]*, save (optional)*[boolean]*) [static]

Description

Map a window allowing you to input a filename (or select it using a file selector). OK and Cancel buttons are shown. See also Window.GetFile().

Arguments

- **title** (string)

Title for window.

- **message** (string)

Message to show in window.

- **extension (optional)** (string)

Extension to filter by.

- **initial (optional)** (string)

Initial value.

- **save (optional)** (boolean)

If true the file selector is to be used for saving a file. If false (default) the file selector is for opening a file. Due to native

operating system file selector differences, on linux new filenames can only be given when saving a file. On windows it is possible to give new filenames when opening or saving a file.

Returns

filename (string), (or null if cancel pressed).

Return type

String

Example

To create an file input window with title *Choose file* and message *Choose the file to open* and return the filename input:

```
var filename = Window.GetFilename("Choose file", "Choose the file to open");
```

GetFiles(extension (optional)*[string]*) [static]

Description

Map a file selector box allowing you to choose multiple files. See also Window.GetFile() and Window.GetFilename().

Arguments

- **extension (optional)** (string)

Extension to filter by.

Returns

Array of filenames (strings), or null if cancel pressed.

Return type

String

Example

To select multiple files using extension '.key':

```
var files = Window.GetFiles(".key");
```

GetGraphicsWindowPosition() [static]

Description

This function returns the current position of the graphics window.

Arguments

No arguments

Returns

Array of numbers containing the left, right, bottom and top positions (in the range 0.0 to 1.0)

Return type

Number

Example

To get the current position of the graphics window:

```
var pos = Window.GetGraphicsWindowPosition();
var l = pos[0];
var r = pos[1];
var b = pos[2];
var t = pos[3];
```

GetInteger(title*[string]*, message*[string]*, initial (optional)*[integer]*) [static]

Description

Map a window allowing you to input an integer. OK and Cancel buttons are shown.

Arguments

- **title** (string)

Title for window.

- **message** (string)

Message to show in window.

- **initial (optional)** (integer)

Initial value.

Returns

value input (integer), or null if cancel pressed.

Return type

Number

Example

To create an input window with title *Input* and message *Input integer* and return the value input:

```
var value = Window.GetInteger("Input", "Input integer");
```

GetNumber(title*[string]*, message*[string]*, initial (optional)*[real]*) [static]

Description

Map a window allowing you to input a number. OK and Cancel buttons are shown.

Arguments

- **title** (string)

Title for window.

- **message** (string)

Message to show in window.

- **initial (optional)** (real)

Initial value.

Returns

value input (real), or null if cancel pressed.

Return type

Number

Example

To create an input window with title *Input* and message *Input number* and return the value input:

```
var value = Window.GetNumber("Input", "Input number");
```

GetPassword(title*[string]*, message*[string]*) [static]

Description

Map a window allowing you to input a password. OK and Cancel buttons are shown.
This is identical to Window.GetString except the string is hidden and no initial value can be given.

Arguments

- **title** (string)

Title for window.

- **message** (string)

Message to show in window.

Returns

value input (string), or null if cancel pressed.

Return type

String

Example

To create an input window with title *Input* and message *Input password* and return the value input:

```
var value = Window.GetPassword("Input", "Input password");
```

GetString(title*[string]*, message*[string]*, initial (optional)*[string]*) [static]

Description

Map a window allowing you to input a string. OK and Cancel buttons are shown.

Arguments

- **title** (string)

Title for window.

- **message** (string)

Message to show in window.

- **initial (optional)** (string)

Initial value.

Returns

value input (string), or null if cancel pressed.

Return type

String

Example

To create an input window with title *Input* and message *Input string* and return the value input:

```
var value = Window.GetString("Input", "Input string");
```

Hide()

Description

Hides (unmaps) the window. Also see the Window onHide property.

Arguments

No arguments

Returns

No return value

Example

To hide window w:

```
w.Hide();
```

Information(title*[string]*, info*[string]*, buttons (optional)*[constant]*) [static]

Description

Show information in a window.

Arguments

- **title** (string)

Title for window.

- **info** (string)

Information to show in window. The maximum number of lines that can be shown is controlled by the Options.max_window_lines option.

- **buttons (optional)** (constant)

The buttons to use. Can be bitwise OR of Window.OK, Window.CANCEL, Window.YES or Window.NO. If this is omitted an OK button will be used. By default the window will be modal. If Window.NONMODAL is also given the window will be non-modal instead.

Returns

Button pressed

Return type

Number

Example

To show information *Information* in window with title *Example* with OK and Cancel buttons:

```
var answer = Window.Information("Example", "Information", Window.OK |
Window.CANCEL);
if (answer == Window.CANCEL) Message("You pressed the Cancel button");
```

MasterResolution() [static]

Description

Returns the resolution of the master programme window in pixels

Arguments

No arguments

Returns

Array of numbers containing x and y resolution in pixels

Return type

Number

Example

To get the resolution of the main window:

```
var res = Window.MasterResolution();
```

Message(title*[string]*, message*[string]*, buttons (optional)*[constant]*) [static]

Description

Show a message in a window.

Arguments

- **title** (string)

Title for window.

- **message** (string)

Message to show in window. The maximum number of lines that can be shown is controlled by the Options.max_window_lines option.

- **buttons (optional)** (constant)

The buttons to use. Can be bitwise OR of Window.OK, Window.CANCEL, Window.YES or Window.NO. If this is omitted an OK button will be used By default the window will be modal. If Window.NONMODAL is also given the window will be non-modal instead.

Returns

Button pressed

Return type

Number

Example

To show message *Press YES or NO* in window with title *Example* with YES and NO buttons:

```
var answer = Window.Message("Example", "Press YES or NO", Window.YES |
Window.NO);
if (answer == Window.NO) Message("You pressed No");
```

MiddleBorder() [static]

Description

Returns the vertical position of the middle border (in range 0-1). The middle border is the border between the tools/keywords window and the docked windows. This can be used to help position windows on the screen.

Arguments

No arguments

Returns

real in range 0-1

Return type

Number

Example

To obtain the position of the middle border:

```
var b = Window.MiddleBorder();
```

Question(title*[string]*, question*[string]*, buttons (optional)*[constant]*) [static]

Description

Show a question in a window.

Arguments

- **title** (string)

Title for window.

- **question** (string)

Question to show in window. The maximum number of lines that can be shown is controlled by the Options.max_window_lines option.

- **buttons (optional)** (constant)

The buttons to use. Can be bitwise OR of Window.OK, Window.CANCEL, Window.YES or Window.NO. If this is omitted Yes and No button will be used. By default the window will be modal. If Window.NONMODAL is also given the window will be non-modal instead.

Returns

Button pressed

Return type

Number

Example

To show question *Do you want to continue?* in window with title *Question*:

```
var answer = Window.Question("Question", "Do you want to continue?");
if (answer == Window.NO) Message("You pressed No");
```

Recompute()

Description

Recomputes the positions of widgets in the window. If you have static widgets and 'normal' widgets in a window and you show and/or hide widgets the window needs to be recomputed to refresh the graphics, scroll bars etc. Calling this method will recompute and redraw the window.

Arguments

No arguments

Returns

No return value

Example

To recompute window w:

```
w.Recompute();
```

Redraw()

Description

Redraws the window. Sometimes if you show, hide or draw graphics on widgets the window needs to be redrawn to refresh the graphics. Calling this method will redraw the window refreshing the graphics.

Arguments

No arguments

Returns

No return value

Example

To redraw window w:

```
w.Redraw();
```

RightBorder() [static]

Description

Returns the horizontal position of the right border (in range 0-1). This can be used to help position windows on the screen.

Arguments

No arguments

Returns

real in range 0-1

Return type

Number

Example

To obtain the position of the right border:

```
var b = Window.RightBorder();
```

SetGraphicsWindowPosition(left*[real]*, right*[real]*, bottom*[real]*, top*[real]*) [static]

Description

This function allows you to move or resize the graphics window.

Arguments

- **left** (real)

left coordinate of graphics window in range 0.0 (left) to 1.0 (right)

- **right** (real)

right coordinate of graphics window in range 0.0 (left) to 1.0 (right)

- **bottom** (real)

bottom coordinate of graphics window in range 0.0 (bottom) to 1.0 (top)

- **top** (real)

top coordinate of graphics window in range 0.0 (bottom) to 1.0 (top)

Returns

No return value

Example

To move/resize the graphics window to be in the top left half of the screen:

```
Window.SetGraphicsWindowPosition(0.0, 0.5, 0.5, 1.0);
```

SetGraphicsWindowSize(width*[integer]*, height*[integer]*) [static]

Description

This function allows you to resize the graphics window.

Arguments

- **width** (integer)

Width of the graphics window in pixels

- **height** (integer)

Height of the graphics window in pixels

Returns

No return value

Example

To resize the graphics window to be 400 pixels wide and 300 pixels high:

```
Window.SetGraphicsWindowSize(400, 300);
```

Show(modal (optional)*[boolean]*)

Description

Shows (maps) the window and waits for user input.

Arguments

- **modal (optional)** (boolean)

If this window is modal (true) then the user is blocked from doing anything else in PRIMER until this window is dismissed). If non-modal (false) then the user can still use other functions in PRIMER.
If omitted the window will be modal.
Note that making a window modal will stop interaction in all other windows and may prevent operations such as picking from working in any macros that are run from scripts.
Before version 21 PRIMER would create a blocking event loop when the Window Show method was called (the call to the Window Show method would not return until the window was hidden again). In PRIMER version 21 the logic has been changed so that Window Show maps the window and returns straight away. The window is managed from the main event loop in PRIMER. In most cases this will make no differences to scripts but there may be rare cases where you specifically want the blocking behaviour. In this case use Window.StartLoop().

Returns

No return value

Example

To show window w:

```
w.Show();
```

To show window w allowing the user to use other functions in PRIMER:

```
w.Show(false);
```

StartLoop() [static]

Description

Starts a blocking event loop in PRIMER. Before version 21 PRIMER would create a blocking event loop when the Window Show method was called (the call to the Window Show method would not return until the window was hidden again). In PRIMER version 21 the logic has been changed so that Window Show maps the window and returns straight away. The window is managed from the main event loop in PRIMER. In most cases this will make no differences to scripts but there may be rare cases where you specifically want the blocking behaviour. An example of this is using a keyout hook script, or if you want to give aprompt/question similarly to Window.Message() where you do **not** want to return to the main event loop. You want to block until the user has specified the action.
Window.StartLoop() can be used to start a local blocking, event loop. To terminate the event loop and resume execution of the script use Window.EndLoop()
Note that only a single loop can be active in PRIMER at any one time. i.e. they cannot be nested. Window.StartLoop() should be used as sparingly as possible and only used for specific short events (e.g. something like the equivalent of Window.Message() where you really do need to block) as if it is used in one script it will prevent it from being used in another script.

Arguments

No arguments

Returns

No return value

Example

To start a blocking event loop:

```
Window.StartLoop();
```

Theme(theme (optional)*[constant]*) [static]

Description

Set or get a user interface theme.

Arguments

- **theme (optional)** (constant)

If it is provided it is used to set the current theme. Can be either Window.USE_OLD_UI_JS, Window.THEME_CURRENT, Window.THEME_DARK, Window.THEME_LIGHT, Window.THEME_CLASSIC.

Returns

Integer. When getting the theme one of: Window.USE_OLD_UI_JS, Window.THEME_DARK, Window.THEME_LIGHT, Window.THEME_CLASSIC

Return type

Number

Example

To determine the current theme:

```
var ui = Window.Theme();
      if(ui == Window.THEME_DARK)
      {
          print("Theme is dark\n");
      }
      else if(ui == Window.THEME_LIGHT)
      {
          print("Theme is light\n");
      }
      else if(ui == Window.THEME_CLASSIC)
      {
          print("Theme is classic\n");
      }
      else
      {
          print("Theme is not set\n");
      }
```

To keep the original (pre v17) appearance of your JavaScript (default):

```
Window.Theme(Window.USE_OLD_UI_JS);
```

To use the current user interface theme:

```
Window.Theme(Window.THEME_CURRENT);
```

To use the dark user interface theme:

```
Window.Theme(Window.THEME_DARK);
```

TopBorder() [static]

Description

Returns the vertical position of the top border (in range 0-1). This can be used to help position windows on the screen. This is no longer used in PRIMER and will always be 1 but is left for backwards compatibility.

Arguments

No arguments

Returns

real in range 0-1

Return type

Number

Example

To obtain the position of the top border:

```
var b = Window.TopBorder();
```

UpdateGUI() [static]

Description

Force GUI to be updated. This function is not normally needed but if you are doing a computationally expensive operation and want to update the GUI it may be necessary as the GUI update requests are cached until there is spare time to update them. Calling this function forces any outstanding requests to be flushed.

Arguments

No arguments

Returns

No return value

Example

To force update of GUI:

```
Window.UpdateGUI();
```

Warning(title*[string]*, warning*[string]*, buttons (optional)*[constant]*) [static]

Description

Show a warning message in a window.

Arguments

- **title** (string)

Title for window.

- **warning** (string)

Warning message to show in window. The maximum number of lines that can be shown is controlled by the Options.max_window_lines option.

- **buttons (optional)** (constant)

The buttons to use. Can be bitwise OR of Window.OK, Window.CANCEL, Window.YES or Window.NO. If this is omitted an OK button will be used. By default the window will be modal. If Window.NONMODAL is also given the window will be non-modal instead.

Returns

Button pressed

Return type

Number

Example

To show warning *Title is blank\nSet to ID?* in window with title *Warning* with Yes and No buttons:

```
var answer = Window.Warning("Warning", "Title is blank\nSet to ID?", Window.YES
| Window.NO);
if (answer == Window.NO) Message("You pressed No");
```

Workflow class

The Workflow class allows you to read and write workflow files. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- GetWorkflowDataFromPath(folder*[string]*, workflow_name (optional)*[string]*)
- ModelIdFromIndex(model_index*[integer]*, workflow_name (optional)*[string]*)
- ModelUnitSystemFromIndex(model_index*[integer]*, workflow_name (optional)*[string]*)
- ModelUserDataBuildFromIndex(model_index*[integer]*, workflow_name (optional)*[string]*)
- ModelUserDataFromIndex(model_index*[integer]*, workflow_name (optional)*[string]*)
- ModelUserDataProgramFromIndex(model_index*[integer]*, workflow_name (optional)*[string]*)
- ModelUserDataVersionFromIndex(model_index*[integer]*, workflow_name (optional)*[string]*)
- NumberOfModels(workflow_name (optional)*[string]*)
- Refresh()
- WorkflowDefinitionFilename(workflow_name (optional)*[string]*)
- WriteToFile(user_data*[object]*, output_filename*[string]*, workflow_definition_filename*[string]*, extra (optional)*[object]*)
- WriteToModel(user_data*[object]*, model*[Model]*, workflow_definition_filename*[string]*, extra (optional)*[object]*)

Workflow constants

Constants for UnitSystem

Name	Description
Workflow.UNIT_SYSTEM_NONE	No unit system
Workflow.UNIT_SYSTEM_U1	U1 unit system (m, kg, s)
Workflow.UNIT_SYSTEM_U2	U2 unit system (mm, t, s)
Workflow.UNIT_SYSTEM_U3	U3 unit system (mm, kg, ms)
Workflow.UNIT_SYSTEM_U4	U4 unit system (mm, g, ms)
Workflow.UNIT_SYSTEM_U5	U5 unit system (ft, slug, s)
Workflow.UNIT_SYSTEM_U6	U6 unit system (m, t, s)

Detailed Description

The Workflow class gives you simple functions to read and write workflow files

Details of functions

GetWorkflowDataFromPath(folder*[string]*, workflow_name (optional)*[string]*) [static]

Description

Returns user data in the specified folder. This will first search for user data in *.json files in the specified folder and upwards in parent folders if there is no user data in the specified folder. The number of folders it will search is controlled by the preference oasys*workflow_max_upward_folder_search_depth. Additionally, if the preference oasys*workflow_user_directory_name is defined it will search in the folder with that name too in the specified folder and parent folders. Secondly, it will search for user data in keyword files in the specified folder.

Arguments

- **folder** (string)

The folder to search for user data.

- **workflow_name (optional)** (string)

The workflow name to return the user data for. If not specified it will return user data for all the workflows found.

Returns

Array of objects with the following properties:

Name	Type	Description
data	object	User data as an object
model_unit_system	integer	Model unit system associated with the user data
workflow_data_filename	string	Filename where workflow data was read from
workflow_definition	string	Workflow definition filename
workflow_name	string	Workflow name

Return type

object

ModelIdFromIndex(model_index*[integer]*, workflow_name (optional)*[string]*) [static]

Description

Returns the id of a model that has data for the selected workflow by index (starting at 0).

Arguments

- **model_index** (integer)

The index of the model to return the id for.

- **workflow_name (optional)** (string)

If defined the id of the model that has data for the named workflow is returned. If it is not specified it uses the name of the workflow selected by the user from the workflow menu.

Returns

integer

Return type

Number

Example

To get the id of the first model that has data for the workflow selected by the user

```
var id = Workflow.ModelIdFromIndex(0);
```

To get the id of the first model that has data for the specified workflow "Automotive Assessment"

```
var id = Workflow.ModelIdFromIndex(0, "Automotive Assessment");
```

ModelUnitSystemFromIndex(model_index*[integer]*, workflow_name (optional)*[string]*) [static]

Description

Returns the unit system of a model that has data for the selected workflow by index (starting at 0). Will be Workflow.UNIT_SYSTEM_NONE or Workflow.UNIT_SYSTEM_U1 or Workflow.UNIT_SYSTEM_U2 or Workflow.UNIT_SYSTEM_U3 or Workflow.UNIT_SYSTEM_U4 or Workflow.UNIT_SYSTEM_U5 or Workflow.UNIT_SYSTEM_U6

Arguments

- **model_index** (integer)

The index of the model to return the unit system for.

- **workflow_name (optional)** (string)

The workflow name to return the unit system for. This is required when a PRIMER item is generated from REPORTER. If it is not specified the first workflow unit system associated with the model is returned (this is the ’normal’ case where a user launches a workflow from the workflow menu).

Returns

integer

Return type

Number

Example

To get the unit system of the first model that has data for the workflow selected by the user

```
var unit_system = Workflow.ModelUnitSystemFromIndex(0);
```

To get the unit system of the second model that has data for the "Automotive Assessment" workflow

```
var unit_system = Workflow.ModelUnitSystemFromIndex(1, "Automotive Assessment");
```

ModelUserDataBuildFromIndex(model_index*[integer]*, workflow_name (optional)*[string]*) [static]

Description

Returns the build number of the program that was used to write out the user data of a model for the selected workflow by index (starting at 0).

Arguments

- **model_index** (integer)

The index of the model to return the program build number for.

- **workflow_name (optional)** (string)

The workflow name to return the build number for. This is required when a PRIMER item is generated from REPORTER. If it is not specified the build number for the first user data associated with the model is returned (this is the ’normal’ case where a user launches a workflow from the workflow menu).

Returns

number

Return type

number

Example

To get the build number of the program that was used to write user data for the first model that has data for the workflow selected by the user

```
var build = Workflow.ModelUserDataBuildFromIndex(0);
```

To get the build number of the program that was used to write user data for the second model that has data for the "Automotive Assessment" workflow

```
var build = Workflow.ModelUserDataBuildFromIndex(1, "Automotive Assessment");
```

ModelUserDataFromIndex(model_index*[integer]*, workflow_name (optional)*[string]*) [static]

Description

Returns the user data associated with a model that has data for the selected workflow by index (starting at 0).

Arguments

- **model_index** (integer)

The index of the model to return the user data for.

- **workflow_name (optional)** (string)

The workflow name to return the user data for. If it is not specified it uses the name of the workflow selected by the user from the workflow menu.

Returns

object

Return type

Object

Example

To get the user data for the first model that has data for the workflow selected by the user

```
var user_data = Workflow.ModelUserDataFromIndex(0);
```

To get the user data for the second model that has data for the "Automotive Assessment" workflow

```
var user_data = Workflow.ModelUserDataFromIndex(1, "Automotive Assessment");
```

ModelUserDataProgramFromIndex(model_index*[integer]*, workflow_name (optional)*[string]*) [static]

Description

Returns the name of the program that was used to write out the user data of a model for the selected workflow by index (starting at 0).

Arguments

- **model_index** (integer)

The index of the model to return the program name for.

- **workflow_name (optional)** (string)

The workflow name to return the program name for. This is required when a PRIMER item is generated from REPORTER. If it is not specified the program name for the first user data associated with the model is returned (this is the ’normal’ case where a user launches a workflow from the workflow menu).

Returns

string

Return type

String

Example

To get the name of the program that was used to write user data for the first model that has data for the workflow selected by the user

```
var program = Workflow.ModelUserDataProgramFromIndex(0);
```

To get the name of the program that was used to write user data for the second model that has data for the "Automotive Assessment" workflow

```
var program = Workflow.ModelUserDataProgramFromIndex(1, "Automotive
Assessment");
```

ModelUserDataVersionFromIndex(model_index*[integer]*, workflow_name (optional)*[string]*) [static]

Description

Returns the version of the program that was used to write out the user data of a model for the selected workflow by index (starting at 0).

Arguments

- **model_index** (integer)

The index of the model to return the program version for.

- **workflow_name (optional)** (string)

The workflow name to return the version for. This is required when a PRIMER item is generated from REPORTER. If it is not specified the version for the first user data associated with the model is returned (this is the 'normal' case where a user launches a workflow from the workflow menu).

Returns

number

Return type

number

Example

To get the version of the program that was used to write user data for the first model that has data for the workflow selected by the user

```
var version = Workflow.ModelUserDataVersionFromIndex(0);
```

To get the version of the program that was used to write user data for the second model that has data for the "Automotive Assessment" workflow

```
var version = Workflow.ModelUserDataVersionFromIndex(1, "Automotive
Assessment");
```

NumberOfModels(workflow_name (optional)*[string]*) [static]

Description

Returns the number of models that have data for the workflow selected by the user.

Arguments

- **workflow_name (optional)** (string)

If defined the number of models that have data for the named workflow is returned. If not defined the number of models that have data for the workflow selected by the user on the workflow menu is returned

Returns

integer

Return type

Number

Example

To get the number of models that have data

```
var n = Workflow.NumberOfModels();
```

Refresh() [static]

Description

Scan for fresh workflow data

Arguments

No arguments

Returns

No return value

Example

```
Workflow.Refresh();
```

WorkflowDefinitionFilename(workflow_name (optional)*[string]*) [static]

Description

Returns the workflow definition filename

Arguments

- **workflow_name (optional)** (string)

The workflow name to return the workflow defintion filename for. If it is not specified it uses the name of the workflow selected by the user from the workflow menu.

Returns

string

Return type

String

Example

To get the worklow definition filename that the script has been run with

```
var workflow_definition_filename = Workflow.WorkflowDefinitionFilename();
```

WriteToFile(user_data*[object]*, output_filename*[string]*, workflow_definition_filename*[string]*, extra (optional)*[object]*) [static]

Description

Writes a workflow to a JSON file. If the file already exists the workflow is added to the file (or updated if it is already in the file).

Arguments

- **user_data** (object)

Object containing user data required for the workflow.

- **output_filename** (string)

Filename to write to.

- **workflow_definition_filename** (string)

Filename of the workflow definition file.

- **extra (optional)** (object)

Extra workflow information

Object has the following properties:

Name	Type	Description

model_unit_system (optional)	integer	The model unit system. Can be Workflow.UNIT_SYSTEM_NONE or Workflow.UNIT_SYSTEM_U1 or Workflow.UNIT_SYSTEM_U2 or Workflow.UNIT_SYSTEM_U3 or Workflow.UNIT_SYSTEM_U4 or Workflow.UNIT_SYSTEM_U5 or Workflow.UNIT_SYSTEM_U6

Returns

No return value

Example

To write the file "C:\my_workflow.json" with some user data and the contents of the workflow definition file "C:\workflows\my_workflow_definition.json"

```
      var user_data = { part_ids: [1, 2, 10, 100], node_ids: [12, 23, 24] };
      Workflow.WriteToFile(user_data, "C:\\my_workflow.json",
"C:\\workflows\\my_workflow_definition.json");
```

WriteToModel(user_data*[object]*, model*[Model]*, workflow_definition_filename*[string]*, extra (optional)*[object]*) [static]

Description

Writes a workflow to a model (updating it if it already exists)

Arguments

- **user_data** (object)

Object containing user data required for the workflow.

- **model** (Model)

Model to write to.

- **workflow_definition_filename** (string)

Filename of the workflow definition file.

- **extra (optional)** (object)

Extra workflow information

Object has the following properties:

Name	Type	Description
model_unit_system (optional)	integer	The model unit system. Can be Workflow.UNIT_SYSTEM_NONE or Workflow.UNIT_SYSTEM_U1 or Workflow.UNIT_SYSTEM_U2 or Workflow.UNIT_SYSTEM_U3 or Workflow.UNIT_SYSTEM_U4 or Workflow.UNIT_SYSTEM_U5 or Workflow.UNIT_SYSTEM_U6

Returns

No return value

Example

To write the workflow to the first model with some user data and the contents of the workflow definition file "C:\workflows\my_workflow_definition.json"

```
      var user_data = { part_ids: [1, 2, 10, 100], node_ids: [12, 23, 24] };
      var m = Model.First();
      Workflow.WriteToModel(user_data, m, "C:\\workflows\\my_workflow_
definition.json");
```

XlsxWorkbook class

The XlsxWorkbook class enables writing xlsx files. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Member functions

- Close()

XlsxWorkbook properties

Name	Type	Description
filename (read only)	string	Name of the xlsx file

Detailed Description

The XlsxWorkbook class provides functions to enable you to create xlsx files. The following simple example shows how to write a xlsx file:

```
 var workbook = new XlsxWorkbook("C:/temp/test.xlsx");
var worksheet = new XlsxWorksheet(workbook);
worksheet.AddText(0, 0, "Hello world!");
worksheet.AddNumber(1, 0, 1.2345);
worksheet.AddImage(2, 0, "image.png");
workbook.Close();
```

Constructor

new XlsxWorkbook(filename*[string]*)

Description

Create a new XlsxWorkbook object for writing xlsx files.

Arguments

- **filename** (string)

Filename of the xlsx file you want to write. The file will be overwritten (if it exists).

Returns

XlsxWorkbook object

Return type

XlsxWorkbook

Example

To create a new XlsxWorkbook object to write Xlsx file "/data/test/file.xlsx"

```
var workbook = new XlsxWorkbook("/data/test/file.xlsx");
```

Details of functions

Close()

Description

Close a Xlsx file

Arguments

No arguments

Returns

No return value

Example

To close Xlsx file workbook:

```
workbook.Close();
```

XlsxWorksheet class

Member functions

- AddImage(row*[integer]*, column*[integer]*, filename*[string]*, x_scale (optional)*[real]*, y_scale (optional)*[real]*)
- AddNumber(row*[integer]*, column*[integer]*, value*[number]*)
- AddText(row*[integer]*, column*[integer]*, text*[string]*)
- SetColumnProperties(column*[integer]*, width*[number]*)
- SetRowProperties(row*[integer]*, height*[number]*)

Constructor

new XlsxWorksheet(workbook*[XlsxWorkbook object]*, name (optional)*[string]*)

Description

Create a new XlsxWorksheet object for writing xlsx files.

Arguments

- **workbook** (XlsxWorkbook object)

The workbook to create the worksheet in.

- **name (optional)** (string)

The name of the worksheet. If omitted the default names 'Sheet1', 'Sheet2' etc will be used.

Returns

XlsxWorksheet object

Return type

XlsxWorksheet

Example

To create a new worksheet in workbook

```
var worksheet = new XlsxWorksheet(workbook);
```

Details of functions

AddImage(row*[integer]*, column*[integer]*, filename*[string]*, x_scale (optional)*[real]*, y_scale (optional)*[real]*)

Description

Add an image to the Xlsx file. Note that the image will not actually be read/inserted until the workbook is written by calling XlsxWorkbook.Close so you must make sure the image file exists until then.

Arguments

- **row** (integer)

The row in the xlsx file (rows start at zero)

- **column** (integer)

The column in the xlsx file (columns start at zero)

- **filename** (string)

Name of the image file you want to add to the xlsx file. The image can be in png or jpeg format.

- **x_scale (optional)** (real)

Scale factor for horizontally scaling the input image.

- **y_scale (optional)** (real)

Scale factor for vertically scaling the input image.

Returns

No return value

Example

To add image 'C:/temp/test.png' to XlsxWorksheet worksheet on the second row, third column:

```
worksheet.AddImage(1, 2, 'C:/temp/test.png');
```

AddNumber(row*[integer]*, column*[integer]*, value*[number]*)

Description

Add number to the Xlsx file

Arguments

- **row** (integer)

The row in the xlsx file (rows start at zero)

- **column** (integer)

The column in the xlsx file (columns start at zero)

- **value** (number)

Number you want to add to the xlsx file

Returns

No return value

Example

To add number 1.2345 to XlsxWorksheet worksheet on the second row, third column:

```
worksheet.AddNumber(1, 2, 1.2345);
```

AddText(row*[integer]*, column*[integer]*, text*[string]*)

Description

Add text to the Xlsx file

Arguments

- **row** (integer)

The row in the xlsx file (rows start at zero)

- **column** (integer)

The column in the xlsx file (columns start at zero)

- **text** (string)

Text you want to add to the xlsx file

Returns

No return value

Example

To add text ’test’ to XlsxWorksheet worksheet on the second row, third column:

```
worksheet.AddText(1, 2, ’test’);
```

SetColumnProperties(column*[integer]*, width*[number]*)

Description

Set the column properties in the worksheet

Arguments

- **column** (integer)

The column in the xlsx file (columns start at zero)

- **width** (number)

Width of the column to set

Returns

No return value

Example

To set the width of the third column in XlsxWorksheet worksheet to 30:

```
worksheet.SetColumnProperties(2, 30);
```

SetRowProperties(row*[integer]*, height*[number]*)

Description

Set the row properties in the worksheet

Arguments

- **row** (integer)

The row in the xlsx file (rows start at zero)

- **height** (number)

Height of the row to set

Returns

No return value

Example

To set the height of the third row in XlsxWorksheet worksheet to 20:

```
worksheet.SetRowProperties(2, 20);
```

XMLParser class

The XMLParser class enables reading data from XML files. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Member functions

- Parse(filename*[string]*)

XMLParser properties

Name	Type	Description
characterDataHandler	function	Function to call when character data is found. The function will be called with 1 argument which is a string containing the character data
commentHandler	function	Function to call when a comment is found. The function will be called with 1 argument which is a string containing the text inside the comment
endCDATAHandler	function	Function to call at the end of a CDATA section. The function does not have any arguments.
endElementHandler	function	Function to call when an element end tag is found. The function will be called with 1 argument which is a string containing the name of the element
startCDATAHandler	function	Function to call at the start of a CDATA section. The function does not have any arguments.
startElementHandler	function	Function to call when an element start tag is found. The function will be called with 2 arguments. Argument 1 is a string containing the name of the element. Argument 2 is an object containing the element attributes

Detailed Description

The XMLParser class provides a stream-oriented parser to enable you to read XML files. You register callback (or handler) functions with the parser and then parse the document. As the parser recognizes parts of the document, it will call the appropriate handler for that part (if you've registered one.) The document is fed to the parser in pieces. This allows you to parse really huge documents that won't fit into memory.
There are currently 6 handlers which can be set: XMLParser.startElementHandler, XMLParser.endElementHandler, XMLParser.characterDataHandler, XMLParser.commentHandler, XMLParser.startCDATAHandler and XMLParser.endCDATAHandler.
The following simple example shows how the parser could be used.

```
// Create a new parser object
var p = new XMLParser();
// assign handlers
p.startElementHandler  = startElem;
p.endElementHandler    = endElem;
p.characterDataHandler = text;
p.commentHandler       = comment;
// parse the file
p.Parse("/data/test.xml");
////////////////////////////////////////////////////////////////////////
function startElem(name, attr)
{
// handler to be called when a start element is found
// Print element name
```

```
    Println("START: " + name);
// Print attributes
    for (n in attr)
    {
        Println(" attr: " + n + "=" + attr[n]);
    }
}
function endElem(name)
{
// handler to be called when an end element is found
// Print element name
    Println("END: " + name);
}
function text(str)
{
// handler to be called when text is found
// Print text
    Println("TEXT: '" + str + "'");
}
function comment(str)
{
// handler to be called when a comment is found
// Print comment
    Println("COMMENT: '" + str + "'");
}
```

See the documentation below for more details.

Constructor

new XMLParser()

Description

Create a new XMLParser object for reading XML files.

Arguments

No arguments

Returns

XMLParser object

Return type

XMLParser

Example

To create a new XMLParser object

```
var p = new XMLParser();
```

Details of functions

Parse(filename*[string]*)

Description

starts parsing an XML file

Arguments

- **filename** (string)

XML file to parse

Returns

No return value

Example

To parse XML file "/data/test.xml"

```
var p = new XMLParser();
p.Parse("/data/test.xml");
```

Xrefs class

The Xrefs class gives you access to cross references. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Member functions

- GetItemID(type*[string]*, pos*[integer]*)
- GetItemType(type*[string]*, pos*[integer]*)
- GetTotal(type*[string]*)
- GetType(n*[integer]*)

Xrefs properties

Name	Type	Description
numtypes (read only)	integer	The number of different types that this item is referenced by.
total (read only)	integer	The total number of cross references of all types to this item.

Detailed Description

The Xrefs class allows you to look at what things use an item. e.g. a node may be used on several shells. See the documentation below for more details.

Details of functions

GetItemID(type*[string]*, pos*[integer]*)

Description

Returns the ID of the item in the reference list.

Arguments

- **type** (string)

The type of the item in the reference list (for a list of types see Appendix I of the PRIMER manual).

- **pos** (integer)

The position in the list for this item. **Note that positions start at 0, not 1**

Returns

ID of item

Return type

Number

Example

To list all of the xrefs for node n:

```
var xrefs = n.Xrefs();
for (var t=0; t<xrefs.numtypes; t++)
{
    var type = xrefs.GetType(t);
    var num  = xrefs.GetTotal(type);
    for (var ref=0; ref<num; ref++)
    {
        var id = xrefs.GetItemID(type, ref);
        Message(type + " " + id + "\n");
    }
}
```

GetItemType(type*[string]*, pos*[integer]*)

Description

Returns the type of the item in the reference list. This function is only required when trying to look at cross references to *DEFINE_CURVE items. These items are used in a slightly different way in PRIMER (each time a curve is used a 'LOADCURVE REFERENCE' structure is created to store things like the units and descriptions of each axis for the curve). If you try to get the cross references for a curve all the references will be of type 'LOADCURVE REFERENCE' and numtypes will be 1. GetItemID() will correctly return the ID of the item from the 'LOADCURVE REFERENCE' structure but to get the type of the item this function is required.

Arguments

- **type** (string)

The type of the item in the reference list (for a list of types see Appendix I of the PRIMER manual).

- **pos** (integer)

The position in the list for this item. **Note that positions start at 0, not 1**

Returns

type of item (String). For every item apart from *DEFINE_CURVE items this will be the same as the *type* argument.

Return type

String

Example

To list all of the xrefs for Curve c:

```
var xrefs = c.Xrefs();
for (var t=0; t<xrefs.numtypes; t++)
{
    var type = xrefs.GetType(t);
    var num  = xrefs.GetTotal(type);
    for (var ref=0; ref<num; ref++)
    {
        var id = xrefs.GetItemID(type, ref);
        var itype = xrefs.GetItemType(type, ref);
        Message(itype + " " + id + "\n");
    }
}
```

GetTotal(type*[string]*)

Description

Returns the total number of references of a type.

Arguments

- **type** (string)

The type of the item in the reference list (for a list of types see Appendix I of the PRIMER manual).

Returns

Number of refs (integer)

Return type

Number

Example

To find the total number of shell references that node n has:

```
var xrefs = n.Xrefs();
    var num  = xrefs.GetTotal("SHELL");
```

GetType(n*[integer]*)

Description

Returns the type for an entry in the reference list. Note that for a curve all the references will be of type 'LOADCURVE REFERENCE' and numtypes will be 1. See GetItemType() for more details.

Arguments

- **n** (integer)

The entry in the reference types that you want the type for. **Note that entries start at 0, not 1**

Returns

The type of the item (string)

Return type

String

Example

To list the types of items that have cross references for node n:

```
var xrefs = n.Xrefs();
for (var t=0; t<xrefs.numtypes; t++)
{
    var type = xrefs.GetType(t);
    var num  = xrefs.GetTotal(type);
    Message(num + " references of type " + type + "\n");
}
```

Zip class

The Zip class enables reading/writing/creating zip files. More...

The PRIMER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Member functions

- AddFile(filename*[string]*, zipname*[string]*)
- Close()
- Information()
- Next()
- ReadFile(filename*[string]*)

Zip constants

Name	Description
Zip.APPEND	Flag to open zip file for appending
Zip.READ	Flag to open zip file for reading
Zip.WRITE	Flag to open zip file for writing

Zip properties

Name	Type	Description
filename (read only)	string	Name of the zip file
mode (read only)	constant	Mode the zip file was opened with (Zip.READ, Zip.WRITE or Zip.APPEND)

Detailed Description

The Zip class provides functions to enable you to read, write and create zip files. The following simple example shows how to write a zip file and then read it again:

```
Message("Creating zip file");
var z = new Zip("C:/temp/test.zip", Zip.WRITE);
z.AddFile("C:/temp/bpost.key",   "bpost/bpost.key");
z.AddFile("C:/temp/door.key",    "door/door.key");
z.AddFile("C:/temp/barrier.key", "other.key");
z.Close();
Message("Done");
var entry = 0;
Message("Reading zip file");
var z = new Zip("C:/temp/test.zip", Zip.READ);
while (true)
{
    entry++;
    Message("Entry "+entry);
    var info = z.Information();
    for (var x in info)
        Message("    "+x+"="+info[x]);
    z.ReadFile(entry+".txt");
    if (!z.Next()) break;
```

```
}
z.Close();
Message("Done")
```

Constructor

new Zip(filename*[string]*, mode*[constant]*)

Description

Create a new Zip object for reading/writing zip files.

Arguments

- **filename** (string)

Filename of the zip file you want to read/write. If reading (Zip.READ) or appending (Zip.APPEND), the file must exist. If writing (Zip.WRITE) the file will be overwritten (if it exists).

- **mode** (constant)

The mode to open the file with. Can be Zip.READ, Zip.WRITE or Zip.APPEND.

Returns

Zip object

Return type

Zip

Example

To create a new Zip object to read Zip file "/data/test/file.zip"

```
var p = new Zip("/data/test/file.zip");
```

Details of functions

AddFile(filename*[string]*, zipname*[string]*)

Description

Add a file to the Zip file

Arguments

- **filename** (string)

Name of the file you want to add to the zip file

- **zipname** (string)

Name to give the file in the zip file

Returns

No return value

Example

To add file 'C:/temp/test.key' to Zip file z with zip name 'test.key':

```
z.AddFile('C:/temp/test.key', 'test.key');
```

Close()

Description

Close a Zip file

Arguments

No arguments

Returns

No return value

Example

To close Zip file z:

```
z.Close();
```

Information()

Description

Gets information for the current entry in the Zip file such as name, size etc

Arguments

No arguments

Returns

Object with the following properties:

Name	Type	Description
compressedSize	integer	Compressed size
crc	integer	Cyclic redundancy check
name	string	Filename
uncompressedSize	integer	Uncompressed size

Return type

object

Example

To get the information:

```
var info = z.Information();
for (var x in info) Println(x + '=' + info[x]);
```

Next()

Description

Go to the next entry in the Zip file

Arguments

No arguments

Returns

true if there is a next entry, false if there are no more entries

Return type

Boolean

Example

To go to the next entry in zip file z:

```
var next = z.Next();
```

ReadFile(filename*[string]*)

Description

Reads the current entry to a file from the Zip file

Arguments

- **filename** (string)

Name of the file you want to create

Returns

No return value

Example

To read the current entry in Zip file z to a file ’test.key’:

```
z.ReadFile(’test.key’);
```

---

global class

The following pages list global class functions from PRIMER which can be used in D3PLOT.

Most of the D3PLOT functions are also in the global scope, but to make them easier to locate, they are presented in different categories (e.g. Contacts, CutSection, Data) in subsequent sections. They are also listed below in alphabetical order.

Class functions

- AllocateFlag()
- DialogueInput(command*[string]*)
- DialogueInputNoEcho(command*[string]*)
- ErrorMessage(string*[Any valid javascript type]*)
- Execute(data*[object]*)
- Exit(write hook interrupt (optional)*[boolean]*)
- ExitTHisLink()
- GetCurrentDirectory()
- GetInstallDirectory()
- GetPreferenceValue(program*[string]*, name*[string]*)
- GetStartInDirectory()
- Getenv(name*[string]*)
- Message(string*[Any valid javascript type]*)
- MilliSleep(time*[integer]*)
- NumberToString(number*[integer/real]*, width*[integer]*, pref_int (optional)*[boolean]*)
- OpenManual(program*[string]*, page*[string]*)
- Print(string*[Any valid javascript type]*)
- Println(string*[Any valid javascript type]*)
- ReturnFlag(flag*[Flag]*)
- SetCurrentDirectory(directory path*[string]*)
- Sleep(time*[integer]*)
- StartTHisLink()
- System(string*[Any valid javascript type]*)
- Unix()
- WarningMessage(string*[Any valid javascript type]*)
- Windows()

D3PLOT functions

- Blank(type_code*[integer]*, item*[integer or array of integers or string]*, window_id (optional)*[integer]*)
- CreateUbinComponent(component_name*[string]*, component_type*[integer]*, data_type*[integer]*, if_existing*[integer]*, dispose (optional)*[integer]*, location (optional)*[integer or string]*)
- CreateWindow(model_list*[Array of integers|integer]*)
- DeleteUbinComponent(handle*[integer]*)
- DeleteWindow(window_list*[Array of numbers|number]*, dispose_flag (optional)*[integer]*)
- GetConditionParts_#deprecated(component*[integer]*, value*[real]*, mode*[integer]*, int_pt (optional)*[object|integer]*, extra (optional)*[integer]*) [deprecated]
- GetContourLimit(mode*[integer]*, component (optional)*[string]*)
- GetCutCoords(options*[object]*)
- GetCutForces(options*[object]*)
- GetCutSection(options*[object]*)
- GetData(component*[integer]*, type_code*[integer]*, item*[integer]*, int_pt (optional)*[object|integer]*, extra (optional)*[integer]*, fr_of_ref (optional)*[integer]*, state_id (optional)*[integer]*, dda (optional)*[integer]*, consider_blanking (optional)*[integer]*, mag_or_cur (optional)*[integer]*)
- GetElemAxes(type_code*[integer]*, item*[integer]*, state_id (optional)*[integer]*)
- GetElemBetaAngle(type_code*[integer]*, item*[integer]*, ply_id*[integer]*, int_pnt (optional)*[integer]*, state_id (optional)*[integer]*)
- GetElemsAtNode(node*[integer]*, type_code*[integer]*, state_id (optional)*[integer]*)
- GetElemsInPart(part_id*[integer]*, state_id (optional)*[integer]*)
- GetElemsInPly(ply_id*[integer]*, state_id (optional)*[integer]*)
- GetGroupInfo(group_id*[integer]*)
- GetIncludeInfo(include_id*[integer]*)
- GetItemsInSet(set_type*[integer]*, set_id*[integer]*)
- GetLabel(type_code*[integer]*, item*[integer]*, state_id (optional)*[integer]*)
- GetMid(type_code*[integer]*, item*[integer]*, layer_id (optional)*[integer]*, state_id (optional)*[integer]*)
- GetModelInfo(model_id (optional)*[integer]*, family_id (optional)*[integer]*)
- GetMultipleData(component*[integer]*, type_code*[integer]*, item_1*[integer]*, item_2*[integer]*, int_pt (optional)*[object|integer]*, extra (optional)*[integer]*, fr_of_ref (optional)*[integer]*, state_id (optional)*[integer]*, dda (optional)*[integer]*, consider_blanking (optional)*[integer]*, mag_or_cur (optional)*[integer]*)

- GetNumOnPlanIntPts(type_code*[integer]*, item*[integer]*, state_id (optional)*[integer]*)
- GetNumberOf(type_code*[integer]*, options (optional)*[object]*)
- GetPartInfo(part_id*[integer]*)
- GetPid(type_code*[integer]*, item*[integer]*, state_id (optional)*[integer]*)
- GetPlyIntPoint(type_code*[integer]*, item*[integer]*, ply_id*[integer]*, state_id (optional)*[integer]*)
- GetPlysInLayup(layup_id*[integer]*, state_id (optional)*[integer]*)
- GetSegmsInSurface(surface_id*[integer]*)
- GetSetInfo(set_type*[integer]*, set_id*[integer]*)
- GetTime(state_id (optional)*[integer]*)
- GetTopology(type_code*[integer]*, item*[integer]*, state_id (optional)*[integer]*)
- GetUbinData(handle*[integer]*, item_type*[integer]*, item*[integer]*, int_pt*[object|integer]*, state_id (optional)*[integer]*)
- GetWindowFrame(window_id*[integer]*)
- GetWindowMaxFrame(window_id*[integer]*)
- GetWindowModels(window_id*[integer]*)
- IsBlanked(type_code*[integer]*, item*[integer]*, window_id (optional)*[integer]*)
- IsDeleted(type_code*[integer]*, item*[integer]*, state_id (optional)*[integer]*)
- IsSelected(type_code*[integer]*, item*[integer]*)
- IsVisible(type_code*[integer]*, item*[integer]*, window_id*[integer]*, state_id (optional)*[integer]*)
- LocateUbinComponent(component_name*[string]*)
- LockState(state_id*[integer]*)
- ModelExists(model_id*[integer]*)
- NumDeleted(type_code*[integer]*, state_id (optional)*[integer]*)
- Pick(type_code*[integer]*, number*[integer]*)
- PutUbinData(handle*[integer]*, item_type*[integer]*, item*[integer]*, int_pt*[object|integer]*, data*[real|array of reals]*, state_id (optional)*[integer]*)
- QueryDataPresent(component*[integer]*, type_code (optional)*[integer]*)
- Select(type_code*[integer]*)
- SetCurrentModel(model_id*[integer]*)
- SetCurrentState(state_id*[integer]*)
- SetCutSection(options*[object]*)
- SetWindowActive(window_id*[integer]*, active_flag*[integer]*)
- SetWindowFrame(window_id*[integer]*, frame_number*[integer]*)
- SpoolNodesInSurface(surface_id*[integer]*, index*[integer]*, side*[integer]*)
- Unblank(type_code*[integer]*, item*[integer or array of integers or string]*, window_id (optional)*[integer]*)
- UnlockState(state_id*[integer]*)

Details of functions

AllocateFlag() [static]

Description

Allocate a flag for use in the script. See also ReturnFlag(). Once allocated the flag is automatically cleared for all the models currently in D3PLOT.

Arguments

No arguments

Returns

Flag to use

Return type

Flag

Example

To allocate a flag

```
var flag = AllocateFlag();
```

DialogueInput(command*[string]*) [static]

Description

Executes one or more command-line syntax commands. There is no limit to the number of lines that may be specified in a single call. See Dialogue Command Syntax for a full list of command-line commands

The DialogueInputNoEcho variant is identical, except that it suppresses the echo of the commands to the dialogue box.

D3PLOT provides a full command-line syntax as an alternative to graphical user interface commands, and a sequence of such commands may be provided here.

Note that:

- Each call to DialogueInput starts at the top of the D3PLOT command-line "tree", at the D3PLOT_MANAGER>>> prompt
- Each call is autonomous, there is no "memory" of where in the command-line tree previous commands finished.
- However within a single call the current command-line tree is remembered from one line to the next.
- Commands are not case-sensitive, although filenames and titles in command strings are.

Therefore commands which require more than one line of input to complete must be specified in a single call; and it makes sense to group a sequence of related commands together in a single call, although this is not mandatory.

If this succeeds it returns true, otherwise false.

Arguments

- **command** (string)

Command to be executed (as if it had been typed into the dialogue box)

This argument can be repeated if required
Alternatively a single array argument containing the multiple values can be given

Returns

No return value

Example

```
// Blanks all solids.
// Unblanks solids 1 to 10.
// Performs a hidden line plot.
// All commands are echoed to the dialogue box
DialogueInput("BLANK SOLID ALL", "UNBLANK SOLID 1 to 10", "HIDDEN");
```

DialogueInputNoEcho(command*[string]*) [static]

Description

Executes one or more command-line syntax commands. There is no limit to the number of lines that may be specified in a single call. See Dialogue Command Syntax for a full list of command-line commands

This does not echo the commands to the dialogue box.

See DialogueInput for more information.

Arguments

- **command** (string)

Command to be executed (as if it had been typed into the dialogue box)

This argument can be repeated if required
Alternatively a single array argument containing the multiple values can be given

Returns

No return value

Example

```
// Read state 10
// Performed a shaded ("greyscale" in command-line syntax)plot
// Create a JPEG format file "image.jpg"
// Command is not echoed to the dialogue box.
DialogueInputNoEcho("STATE10", "/GREY GO", "/IMAGE jpeg image.jpg");
```

ErrorMessage(string*[Any valid javascript type]*) [static]

Description

Print an error message to the dialogue box **adding a carriage return**.

Arguments

- **string** (Any valid javascript type)

The string/item that you want to print

Returns

No return value

Example

To print the title of model object m as an error to the dialogue box

```
ErrorMessage("The title is " + m.title);
```

Execute(data*[object]*) [static]

Description

Execute a program or script outside D3PLOT and get the standard output and error streams.

Arguments

- **data** (object)

Execute data

Object has the following properties:

Name	Type	Description
arguments (optional)	Array of strings	The arguments to pass to program
program	string	The program you want to run. Note that on Linux this will consider PATH when resolving executable filenames without an absolute path. If you want to run something from the current directory and you do not have '.' in your PATH then you will need to write './something' as the program.

Returns

Object with the following properties:

Name	Type	Description
status	integer	The exit code from the program/script
stderr	string	The standard error output from the program/script
stdout	string	The standard output from the program/script

Return type

object

Example

To run script "example.bat" with arguments "foo" and "bar":

```
var output = Execute( { program: 'example.bat', arguments: [ 'foo', 'bar' ]} );
var text   = output.stdout;
var errors = output.stderr;
var ecode  = output.status;
```

Exit(write hook interrupt (optional)*[boolean]*) [static]

Description

Exit script

Arguments

- **write hook interrupt (optional)** (boolean)

If Exit() is called from a write_hook.js script, the first argument will be processed as in the following: If the argument is provided and set to "true", it is used to interrupt the write out of the model, so that the script exits without anything being written out. An argument value of "false" exits the script and allows the model to be written out as normal. An example of this function's use in a Write Hook script can be found at $OA_INSTALL/primer_ library/scripts/hooks/example_write_hook.js.

Returns

No return value

Example

Exit with

```
Exit();
```

ExitTHisLink() [static]

Description

Exits the T/HIS link from D3PLOT

Arguments

No arguments

Returns

No return value

GetCurrentDirectory() [static]

Description

Get the current working directory

Arguments

No arguments

Returns

String containing current working directory

Return type

String

Example

To get the current directory:

```
var cwd = GetCurrentDirectory();
```

GetInstallDirectory() [static]

Description

Get the directory in which executables are installed. This is the OA_INSTALL environment variable, or if that is not set the directory in which the current executable is installed. Returns NULL if not found

Arguments

No arguments

Returns

string

Return type

String

Example

To get the install directory:

```
var install_dir = GetInstallDirectory();
```

GetPreferenceValue(program*[string]*, name*[string]*) [static]

Description

Get the Preference value with the given string in the any of admin ("OA_ADMIN") or install ("OA_INSTALL") or home ("OA_HOME") directory oa_pref

Arguments

- **program** (string)

The program name string : Valid values are ’All’, ’D3PLOT’, ’PRIMER’, ’REPORTER’, ’SHELL’, ’T/HIS’

- **name** (string)

The preference name string

Returns

: String containing preference value or null if preference string is not present in any oa_pref. Also if none of the above environment variables are not present, then API simply returns null. While returning preference value, locked preference value in admin and then install oa_pref takes precedence over home oa_pref. If preference is not locked in any of these oa_pref, preference in home directory oa_pref is returned.

Return type

String

Example

To get the preference value:

```
var pref_list = GetPreferenceValue('All', "font_size");
```

GetStartInDirectory() [static]

Description

Get the directory passed to D3PLOT by the -start_in command line argument

Arguments

No arguments

Returns

String containing start_in directory or NULL if not set

Return type

String

Example

To get the start_in directory:

```
var start_in = GetStartInDirectory();
```

Getenv(name*[string]*) [static]

Description

Get the value of an environment variable

Arguments

- **name** (string)

The environment variable name

Returns

String containing variable value or null if variable does not exist

Return type

String

Example

To get the value for environment variable HOME

```
var home = Getenv("HOME");
```

Message(string*[Any valid javascript type]*) [static]

Description

Print a message to the dialogue box **adding a carriage return**.

Arguments

- **string** (Any valid javascript type)

The string/item that you want to print. If '\r' is added to the end of the string then instead of automatically adding a carriage return in the dialogue box, the next message will overwrite the current one. This may be useful for giving feedback to the dialogue box when doing an operation.

Returns

No return value

Example

To print the title of model object m as a message to the dialogue box

```
Message("The title is " + m.title);
```

MilliSleep(time*[integer]*) [static]

Description

Pause execution of the script for *time* milliseconds. See also Sleep()

Arguments

- **time** (integer)

Number of milliseconds to pause for

Returns

No return value

Example

To pause for 500 milliseconds

```
MilliSleep(500);
```

NumberToString(number*[integer/real]*, width*[integer]*, pref_int (optional)*[boolean]*) [static]

Description

Formats a number to a string with the specified width.

Arguments

- **number** (integer/real)

The number you want to format.

- **width** (integer)

The width of the string you want to format it to (must be less than 80).

- **pref_int (optional)** (boolean)

By default only integer values inside the single precision 32 bit signed integer limit of approximately +/-2e9 are formatted as integers, all other numeric values are formatted as floats. With this argument set to TRUE then integer values up to the mantissa precision of a 64 bit float, approximately +/-9e15, will also be formatted as integers.

Returns

String containing the number

Return type

String

Example

To write the number 1.2345e+6 to a string 10 characters wide

```
var str = NumberToString(1.2345e+6, 10);
```

OpenManual(program*[string]*, page*[string]*) [static]

Description

Open the Oasys manuals at a requested page

Arguments

- **program** (string)

The program manual to open. Can be "primer", "d3plot" or "this"

- **page** (string)

The page to open in the manual, e.g. "running-this.html"

Returns

true if successful, false if not

Return type

Boolean

Example

To open the T/HIS manual on the running-this.html page

```
OpenManual("this", "running-this.html");
```

Print(string*[Any valid javascript type]*) [static]

Description

Print a string to stdout. **Note that a carriage return is not added**.

Arguments

- **string** (Any valid javascript type)

The string/item that you want to print

Returns

No return value

Example

To print string "Hello, world!"

```
Print("Hello, world!");
```

To print the title of model object m with a carriage return

```
print("The title is " + m.title + "\n");
```

Println(string*[Any valid javascript type]*) [static]

Description

Print a string to stdout **adding a carriage return**.

Arguments

- **string** (Any valid javascript type)

The string/item that you want to print

Returns

No return value

Example

To print string "Hello, world!" automatically adding a carriage return

```
Println("Hello, world!");
```

To print the title of model object m, automatically adding a carriage return

```
Println("The title is " + m.title);
```

ReturnFlag(flag*[Flag]*) [static]

Description

Return a flag used in the script. See also AllocateFlag().

Arguments

- **flag** (Flag)

The flag to return

Returns

No return value

Example

To return flag f:

```
ReturnFlag(f);
```

SetCurrentDirectory(directory path*[string]*) [static]

Description

Sets the current working directory.

Arguments

- **directory path** (string)

Path to the directory you would like to change into.

Returns

true if successful, false if not

Return type

Boolean

Example

To change into the directory "/data/test" exists

```
SetCurrentDirectory("/data/test")
```

Sleep(time*[integer]*) [static]

Description

Pause execution of the script for *time* seconds. See also MilliSleep()

Arguments

- **time** (integer)

Number of seconds to pause for

Returns

No return value

Example

To pause for 2 seconds

```
Sleep(2);
```

StartTHisLink() [static]

Description

Starts the T/HIS link from D3PLOT

Arguments

No arguments

Returns

No return value

System(string*[Any valid javascript type]*) [static]

Description

Do a system command outside D3PLOT. To run an external command and get the output then please use Execute() instead.

Arguments

- **string** (Any valid javascript type)

The system command that you want to do

Returns

integer (probably zero if command successful but is implementation-dependant)

Return type

Number

Example

To make the directory "example"

```
System("mkdir example");
```

Unix() [static]

Description

Test whether script is running on a Unix/Linux operating system. See also Windows()

Arguments

No arguments

Returns

true if Unix/Linux, false if not

Return type

Boolean

Example

To test if the OS is Unix

```
if ( Unix() )
```

WarningMessage(string*[Any valid javascript type]*) [static]

Description

Print a warning message to the dialogue box **adding a carriage return**.

Arguments

- **string** (Any valid javascript type)

The string/item that you want to print

Returns

No return value

Example

To print the title of model object m as a warning to the dialogue box

```
WarningMessage("The title is " + m.title);
```

Windows() [static]

Description

Test whether script is running on a Windows operating system. See also Unix()

Arguments

No arguments

Returns

true if Windows, false if not

Return type

Boolean

Example

To test if the OS is Windows

```
if ( Windows() )
```

Beam class

The Beam class gives you access to beam elements in D3PLOT. More...

The D3PLOT JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(window*[GraphicsWindow]*, model*[Model]*)
- BlankFlagged(window*[GraphicsWindow]*, model*[Model]*, flag*[Flag]*)
- First(model*[Model]*)
- FlagAll(model*[Model]*, flag*[Flag]*)
- GetAll(model*[Model]*, property (optional)*[string]*)
- GetFlagged(model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(model*[Model]*, label*[integer]*)
- GetFromIndex(model*[Model]*, index*[integer]*)
- GetMultipleData(component*[constant]*, items*[array]*, options (optional)*[object]*)
- Last(model*[Model]*)
- Pick()
- Select(flag*[Flag]*)
- Total(model*[Model]*)
- TotalDeleted(model*[Model]*)
- UnblankAll(window*[GraphicsWindow]*, model*[Model]*)
- UnblankFlagged(window*[GraphicsWindow]*, model*[Model]*, flag*[Flag]*)
- UnflagAll(model*[Model]*, flag*[Flag]*)

Member functions

- Blank(window*[GraphicsWindow]*)
- Blanked(window*[GraphicsWindow]*)
- ClearFlag(flag*[Flag]*)
- Deleted()
- Flagged(flag*[Flag]*)
- ForceMoment(options (optional)*[object]*)
- GetData(component*[constant]*, options (optional)*[object]*)
- LocalAxes()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Topology()
- Unblank(window*[GraphicsWindow]*)

Beam properties

Name	Type	Description
colour	Colour	The colour of the beam
data (read only)	real|array	Component data for a beam passed as an argument to GetMultipleData. Note that data will only exist for the instance of the beam passed to GetMultipleData. i.e. it is a local property stored on the specific instance. It is not stored in the D3PLOT database
displayMode	integer	The display mode of the beam. Can be Constant.WIREFRAME, Constant.HIDDEN, Constant.SHADED or Constant.CURRENT.
include (read only)	integer	The include file number in the model that the beam is in
index (read only)	integer	The internal index for the beam in D3PLOT (starting at 0)

integrationPoints (read only)	integer	The number of integration points that the beam has
label (read only)	integer	The Ansys LS-DYNA label for the beam
material (read only)	Material	The Material the beam has. This is only available if there is a ztf file for the model. If not null will be returned. If this is a PART_COMPOSITE then null will be returned. If the model comes from writing a PTF/d3plot file from PRIMER and the material was latent there, null will be returned. Part.GetCompositeData should be used to get material data in this case
model (read only)	Model	The Model that the beam is in
overlayColour	Colour	The overlay colour of the beam
overlayMode	integer	The overlay mode of the beam. Can be Constant.NO_OVERLAY, Constant.FREE_EDGE, Constant.ALL_EDGES or Constant.CURRENT.
part (read only)	Part	The Part the beam is in
transparency	integer	The transparency of the beam between 0 (opaque) and 100 (clear)
type (read only)	constant	The type for the beam (will be Type.BEAM)

Detailed Description

The Beam class allows you to inspect beam elements in a model. See the documentation below for more details.

Details of functions

Blank(window*[GraphicsWindow]*)

Description

Blanks the beam in a graphics window

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to blank the beam in

Returns

No return value

Example

To blank beam b in graphics window g:

```
b.Blank(g);
```

BlankAll(window*[GraphicsWindow]*, model*[Model]*) [static]

Description

Blanks all of the beams in the model

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to blank the beams in

- **model** (Model)

Model that all the beams will be blanked in

Returns

No return value

Example

To blank all of the beams in model m, in graphics window gw:

```
Beam.BlankAll(gw, m);
```

BlankFlagged(window*[GraphicsWindow]*, model*[Model]*, flag*[Flag]*) [static]

Description

Blanks all of the beams in the model flagged with a defined flag

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to blank the beams in

- **model** (Model)

Model that the flagged beams will be blanked in

- **flag** (Flag)

Flag (see AllocateFlag) set on the beams to blank

Returns

No return value

Example

To blank all of the beams flagged with flag f in model m, in graphics window gw:

```
Beam.BlankFlagged(gw, m, f);
```

Blanked(window*[GraphicsWindow]*)

Description

Checks if the beam is blanked in a graphics window or not

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) in which to check if the beam is blanked

Returns

true if blanked, false if not

Return type

boolean

Example

To check if beam b is blanked in graphics window g:

```
if (b.Blanked(g) ) do_something...
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on a beam

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to clear on the beam

Returns

No return value

Example

To clear flag f on beam b:

```
b.ClearFlag();
```

Deleted()

Description

Checks if the beam has been deleted or not

Arguments

No arguments

Returns

true if deleted, false if not

Return type

boolean

Example

To check if beam b has been deleted:

```
if (b.Deleted() ) do_something...
```

First(model*[Model]*) [static]

Description

Returns the first beam in the model (or null if there are no beams in the model)

Arguments

- **model** (Model)

Model to get first beam in

Returns

Beam object

Return type

Beam

Example

To get the first beam in model m:

```
var b = Beam.First(m);
```

FlagAll(model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the beams in the model with a defined flag

Arguments

- **model** (Model)

Model that all the beams will be flagged in

- **flag** (Flag)

Flag (see AllocateFlag) to set on the beams

Returns

No return value

Example

To flag all of the beams with flag f in model m:

```
Beam.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the beam is flagged or not

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to test on the beam

Returns

true if flagged, false if not

Return type

boolean

Example

To check if beam b has flag f set on it:

```
if (b.Flagged(f) ) do_something...
```

ForceMoment(options (optional)*[object]*)

Description

Returns the forces and moments for the beam

Arguments

- **options (optional)** (object)

Object containing options for getting data. Can be any of:

Object has the following properties:

Name	Type	Description
ip	integer	Integration point number to get the data at (ip >= 1)

Returns

Array containing the forces and moments [Fx, Fy, Fz, Mxx, Myy, Mzz] (or null if the value cannot be calculated)

Return type

array

Example

To return the forces and moments of beam b:

```
var fm = b.ForceMoment();
if (fm !== null) do_something...
```

GetAll(model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Beam objects or properties for all of the beams in the model. If the optional property argument is not given then an array of Beam objects is returned. If the property argument is given, that property value for each beam is returned in the array instead of a Beam object

Arguments

- **model** (Model)

Model that all the beams are in

- **property (optional)** (string)

Name for property to get for all beams in the model

Returns

Array of Beam objects or properties

Return type

Array

Example

To get all of the beams in model m:

```
var b = Beam.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'label' for a beam) for each beam in model m:

```
var a = Beam.GetAll(m, 'foo');
```

GetData(component*[constant]*, options (optional)*[object]*)

Description

Returns the value for a data component.
Also see GetMultipleData

Arguments

- **component** (constant)

Component constant to get data for

- **options (optional)** (object)

Object containing options for getting data. Can be any of:

Object has the following properties:

Name	Type	Description
extra	integer	The extra data component number if component Component.SOX for solids, Component.BMX for beams or Component.SHX for shells and thick shells
ip	integer	Integration point number to get the data at (ip >= 1 or one of the constants Constant.TOP, Constant.MIDDLE or Constant.BOTTOM). If the integration point is not defined it will use the integration point defined on the current GUI "data" panel, which defaults to the middle surface for shells, thick shells, and solids, and Mag All for beams, but may vary if changed by an interactive user. If consistent output from a script is required, independent of any prior interactive activity, an explicit integration point or surface should be defined
op	integer	On plane integration point number for shells and thick shells (op >= 1 [default])
referenceFrame	constant	The frame of reference to return values in. Either Constant.GLOBAL (default), Constant.LOCAL, Constant.CYLINDRICAL, Constant.USER_DEFINED or Constant.MATERIAL. This is only necessary for directional components (eg X stress) and then only when something other than the default Constant.GLOBAL coordinate system is to be used
user	integer	The user-defined component number if component Component.UNOS, Component.UNOV, Component.USSS, Component.USST, Component.UBMS or Component.UBMV

Returns

Number if a scalar component, array if a vector or tensor component (or null if the value cannot be calculated because it’s not available in the model).
If requesting an invalid component it will throw an error (e.g. Component.AREA of a node).

Return type

real|array

Example

To calculate a component and check it has been calculated (note that in the example, the argument extra is optional):

```
var value = b.GetData(component, {extra: 1});
if (value !== null) do_something...
```

GetFlagged(model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Gets all of the beams in the model flagged with a defined flag. If the optional property argument is not given then an array of Beam objects is returned. If the property argument is given, that property value for each beam is returned in the array instead of a Beam object

Arguments

- **model** (Model)

Model that the flagged beams are in

- **flag** (Flag)

Flag (see AllocateFlag) set on the beams to get

- **property (optional)** (string)

Name for property to get for all flagged beams in the model

Returns

Array of Beam objects or properties

Return type

Array

Example

To get all of the beams flagged with flag f in model m:

```
Beam.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'label' for a node) for all beams flagged with flag f in model m:

```
var a = Beam.GetFlagged(m, f, 'foo');
```

GetFromID(model*[Model]*, label*[integer]*) [static]

Description

Returns the Beam object for beam in model with label (or null if it does not exist)

Arguments

- **model** (Model)

Model to get beam in

- **label** (integer)

The Ansys LS-DYNA label for the beam in the model

Returns

Beam object

Return type

Beam

Example

To get the beam in model m with label 1000:

```
var b = Beam.GetFromID(m, 1000);
```

GetFromIndex(model*[Model]*, index*[integer]*) [static]

Description

Returns the Beam object for beam in model with index (or null if it does not exist)

Arguments

- **model** (Model)

Model to get beam in

- **index** (integer)

The D3PLOT internal index in the model for beam, starting at 0.

Returns

Beam object

Return type

Beam

Example

To get the 51st beam in model m:

```
var b = Beam.GetFromIndex(m, 50);
```

GetMultipleData(component*[constant]*, items*[array]*, options (optional)*[object]*) [static]

Description

Returns the value for a data component for multiple beams. For each beam a local property called data will be created containing a number if a scalar component, or an array if a vector or tensor component (or null if the value cannot be calculated). The data is also returned as an object.
Also see GetData

Arguments

- **component** (constant)

Component constant to get data for

- **items** (array)

Array of Beam objects to get the data for. All of the beams must be in the same model.

- **options (optional)** (object)

Object containing options for getting data. Can be any of:

Object has the following properties:

Name	Type	Description
extra	integer	The extra data component number if component Component.SOX for solids, Component.BMX for beams or Component.SHX for shells and thick shells
ip	integer	Integration point number to get the data at (ip >= 1 or one of the constants Constant.TOP, Constant.MIDDLE or Constant.BOTTOM)
op	integer	On plane integration point number for shells and thick shells (op >= 1 [default])
referenceFrame	constant	The frame of reference to return values in. Either Constant.GLOBAL (default), Constant.LOCAL, Constant.CYLINDRICAL, Constant.USER_DEFINED or Constant.MATERIAL. This is only necessary for directional components (eg X stress) and then only when something other than the default Constant.GLOBAL coordinate system is to be used
user	integer	The user-defined component number if component Component.UNOS, Component.UNOV, Component.USSS, Component.USST, Component.UBMS or Component.UBMV

Returns

Object containing the data. A property is created in the object for each beam with the label. The value of the property is a number if a scalar component or an array if a vector or tensor component (or null if the value cannot be calculated)

Return type

object

Example

To calculate a component for beams in array items and use the data property (note that in the example, the argument extra is optional):

```
Beam.GetMultipleData(component, items, {extra: 1});
for (i=0; i<items.length; i++)
{
    if (items[i].data !== null) do_something...
}
```

To calculate a component for beams in array items and use the return value (note that in the example, the argument extra is optional):

```
var data = Beam.GetMultipleData(component, items, {extra: 1});
for (d in data)
{
    Message("Label is " + d);
    if (data[d] !== null) do_something...
}
```

Last(model[*Model*]) [static]

Description

Returns the last beam in the model (or null if there are no beams in the model)

Arguments

- **model** (Model)

Model to get last beam in

Returns

Beam object

Return type

Beam

Example

To get the last beam in model m:

```
var b = Beam.Last(m);
```

LocalAxes()

Description

Returns the local axes of the element in model space, expressed as direction cosines in a 2D array. Beam elements must have 3 nodes to be able to return local axes.

Arguments

No arguments

Returns

array of arrays

Return type

Array

Example

To get the local axes for beam b:

```
var axes = b.LocalAxes();
var xAxis = [ axes[0][0], axes[0][1], axes[0][2] ];
var yAxis = [ axes[1][0], axes[1][1], axes[1][2] ];
var zAxis = [ axes[2][0], axes[2][1], axes[2][2] ];
```

Next()

Description

Returns the next beam in the model (or null if there is not one)

Arguments

No arguments

Returns

Beam object

Return type

Beam

Example

To get the next beam after beam b:

```
b = b.Next();
```

Pick() [static]

Description

Allows the user to pick a beam from the screen

Arguments

No arguments

Returns

Beam object or null if cancelled

Return type

Beam

Example

To pick a beam:

```
var b = Beam.Pick();
```

Previous()

Description

Returns the previous beam in the model (or null if there is not one)

Arguments

No arguments

Returns

Beam object

Return type

Beam

Example

To get the previous beam before beam b:

```
b = b.Previous();
```

Select(flag*[Flag]*) [static]

Description

Selects beams using an object menu

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to use when selecting beams

Returns

The number of beams selected or null if menu cancelled

Return type

integer

Example

To select beams, flagging those selected with flag f:

```
var total = Beam.Select(f);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on a beam

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to set on the beam

Returns

No return value

Example

To set flag f on beam b:

```
b.SetFlag(f);
```

Topology()

Description

Returns the topology for the beam in the model

Arguments

No arguments

Returns

array of Node objects

Return type

Array

Example

To get the topology for beam b:

```
var topology = b.Topology();
```

Total(model[*Model*]) [static]

Description

Returns the total number of beams in the model

Arguments

- **model** (Model)

Model to get total in

Returns

The number of beams

Return type

integer

Example

To get the number of beams in model m:

```
var total = Beam.Total(m);
```

TotalDeleted(model[*Model*]) [static]

Description

Returns the total number of beams that have been deleted in a model in the state given by its state property

Arguments

- **model** (Model)

Model to get total in

Returns

The number of beams that have been deleted

Return type

integer

Example

To get the number of beams in model m that have been deleted in state 10:

```
m.state = 10;
var total = Beam.TotalDeleted(m);
```

Unblank(window*[GraphicsWindow]*)

Description

Unblanks the beam in a graphics window

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to unblank the beam in

Returns

No return value

Example

To unblank beam b in graphics window g:

```
b.Unblank(g);
```

UnblankAll(window*[GraphicsWindow]*, model*[Model]*) [static]

Description

Unblanks all of the beams in the model

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to unblank the beams in

- **model** (Model)

Model that all the beams will be unblanked in

Returns

No return value

Example

To unblank all of the beams in model m, in graphics window gw:

```
Beam.UnblankAll(gw, m);
```

UnblankFlagged(window*[GraphicsWindow]*, model*[Model]*, flag*[Flag]*) [static]

Description

Unblanks all of the beams in the model flagged with a defined flag

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to unblank the beams in

- **model** (Model)

Model that the flagged beams will be unblanked in

- **flag** (Flag)

Flag (see AllocateFlag) set on the beams to unblank

Returns

No return value

Example

To unblank all of the beams flagged with flag f in model m, in graphics window gw:

```
Beam.UnblankFlagged(gw, m, f);
```

UnflagAll(model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the beams in the model

Arguments

- **model** (Model)

Model that the defined flag for all beams will be unset in

- **flag** (Flag)

Flag (see AllocateFlag) to unset on the beams

Returns

No return value

Example

To unset flag f on all of the beams in model m:

```
Beam.UnflagAll(m, f);
```

Colour class

The Colour class contains constants relating to colours. More...

The D3PLOT JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- GetFromName(name*[string]*)
- RGB(red*[integer]*, green*[integer]*, blue*[integer]*)

Colour constants

Name	Description
Colour.BLACK	Colour black
Colour.BLUE	Colour blue
Colour.CURRENT	Overlay colour defined in display options for objects
Colour.CYAN	Colour cyan
Colour.DARK_ORANGE	Colour dark orange
Colour.DEFAULT	Default colour for objects
Colour.ENTITY	Overlay colour same as entity colour for objects
Colour.GREEN	Colour green
Colour.GREEN_CYAN	Colour green/cyan
Colour.GREY	Colour grey
Colour.LIGHT_BLUE	Colour light blue
Colour.MAGENTA	Colour magenta
Colour.MEDIUM_BLUE	Colour medium blue
Colour.ORANGE	Colour orange
Colour.RED	Colour red
Colour.RED_MAGENTA	Colour red/magenta
Colour.WHITE	Colour white
Colour.YELLOW	Colour yellow
Colour.YELLOW_GREEN	Colour yellow/green

Detailed Description

The Colour class is used to define colours, either by predefined colours or by RGB values. The easiest way to set the colour of something is to use the predefined colour constants. e.g. to set the colour of part p to red:

```
p.colour = Colour.RED;
```

For other colours use Colour.RGB().

Details of functions

GetFromName(name*[string]*) [static]

Description

Returns the colour for a given core or user colour name

Arguments

- **name** (string)

The name of the colour, for example red or user_green or green/cyan.

Returns

colour value (integer)

Return type

Number

RGB(red*[integer]*, green*[integer]*, blue*[integer]*) [static]

Description

Creates a colour from red, green and blue components

Arguments

- **red** (integer)

red component of colour (0-255).

- **green** (integer)

green component of colour (0-255).

- **blue** (integer)

blue component of colour (0-255).

Returns

colour value (integer)

Return type

Number

Component class

The Component class gives you access to user defined binary components in D3PLOT. More...

The D3PLOT JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- First()
- GetFromID(number*[integer]*)
- GetFromName(name*[string]*)
- Last()
- Total()

Member functions

- Delete()
- GetData(item*[Node|Beam|Shell|Solid|Tshell]*, options (optional)*[object]*)
- Next()
- Previous()
- PutData(item*[Node|Beam|Shell|Solid|Tshell]*, data*[real|array]*, options (optional)*[object]*)

Component constants

Constants for ALE Data Components

Name	Description
Component.ADENS	ALE denisty
Component.ADOMF	ALE dominant fraction
Component.AMMG	ALE multi-material group id
Component.AMMS	ALE multi-material group mass

Constants for Basic and Integrated Beam Force/Moment Data Components

Name	Description
Component.BFMV	Force and moment vector [BFX, BFY, BFZ, BMXX, BMYY, BMZZ]
Component.BFR	Force magnitude
Component.BFX	X axial force
Component.BFY	Y axial force
Component.BFZ	Z axial force
Component.BMX	Extra beam data
Component.BMXX	XX torsional moment
Component.BMYY	YY bending moment
Component.BMZZ	ZZ bending moment

Component.BRM	Moment magnitude

Constants for Basic and Integrated Beam Strain Data Components

Name	Description
Component.BEAX	Axial strain
Component.BEP	Effective plastic strain

Constants for Basic and Integrated Beam Stress Data Components

Name	Description
Component.BSXX	Axial stress
Component.BSXY	XY shear stress
Component.BSZX	ZX shear stress

Constants for Belytschko-Schwer Resultant Beam Energy Data Components

Name	Description
Component.BAED	Axial energy density
Component.BAEN	Axial energy
Component.BBED	Bending energy density
Component.BIE	Internal energy
Component.BIED	Internal energy density

Constants for Belytschko-Schwer Resultant Beam Moment Data Components

Name	Description
Component.BMY1	Y moment at end 1
Component.BMY2	Y moment at end 2
Component.BMZ1	Z moment at end 1
Component.BMZ2	Z moment at end 2

Constants for Belytschko-Schwer Resultant Beam Rotation Data Components

Name	Description
Component.BRXX	Torsional rotation
Component.BRY1	Y rotation at end 1
Component.BRY2	Y rotation at end 2
Component.BRZ1	Z rotation at end 1
Component.BRZ2	Z rotation at end 2

Constants for Belytschko-Schwer Resultant Beam Strain Data Components

Name	Description

Component.BPE1	Plastic energy at end 1
Component.BPE2	Plastic energy at end 2
Component.BSAX	Total axial strain

Constants for Contact Surface Data Components (if a .ctf file has been read)

Name	Description
Component.CAREA	Contact segment area
Component.CFGX	Contact global X force
Component.CFGY	Contact global Y force
Component.CFGZ	Contact global Z force
Component.CFLX	Contact local X force
Component.CFLY	Contact local Y force
Component.CFLZ	Contact local Z force
Component.CFM	Contact force magnitude
Component.CSN	Contact normal stress
Component.CST	Contact tangential stress
Component.CSX	Contact local X stress
Component.CSY	Contact local Y stress

Constants for Element Plastic Strain Data Components

Name	Description
Component.EPL	Effective plastic strain
Component.ERATE	Strain rate
Component.PEMAG	Plastic strain magnitude

Constants for Element Plastic Strain Derived Data Components

Name	Description
Component.PEAV	Average plastic strain
Component.PEMAX	Max principal plastic strain
Component.PEMID	Middle principal plastic strain
Component.PEMIN	Min principal plastic strain
Component.PEMS	Max plastic shear strain

Constants for Element Plastic Strain Tensor Data Components

Name	Description
Component.PETEN	Plastic strain tensor [EXX, EYY, EZZ, EXY, EYZ, EZX]
Component.PEXX	X Plastic strain
Component.PEXY	XY Plastic shear strain
Component.PEYY	Y Plastic strain

Component.PEYZ	XY Plastic shear strain
Component.PEZX	ZX Plastic shear strain
Component.PEZZ	Z Plastic strain

Constants for Element Strain Derived Data Components

Name	**Description**
Component.E2MAX	2D (in-plane) max principal strain
Component.E2MIN	2D (in-plane) min principal strain
Component.E2SHEAR	2D (in-plane) max shear strain
Component.EAV	Average strain
Component.EMAX	Max prinicipal strain
Component.EMID	Middle prinicipal strain
Component.EMIN	Min prinicipal strain
Component.EMS	Max shear strain
Component.ENGMAJ	Engineering Major strain
Component.ENGMIN	Engineering Minor strain
Component.ENGTHK	Engineering Thickness strain
Component.ERATIO	2D (in-plane) principal strain ratio
Component.EVON	von Mises strain
Component.SED	Strain energy density

Constants for Element Strain Tensor Data Components

Name	**Description**
Component.ETEN	Strain tensor [EXX, EYY, EZZ, EXY, EYZ, EZX]
Component.EXX	X strain
Component.EXY	XY shear strain
Component.EYY	Y strain
Component.EYZ	YZ shear strain
Component.EZX	ZX shear strain
Component.EZZ	Z strain

Constants for Element Stress Derived Data Components

Name	**Description**
Component.LODE_A	Lode angle
Component.LODE_P	Lode parameter
Component.LODE_PA	Lode parameter alt
Component.S2MAX	2D (in-plane) max principal stress
Component.S2MIN	2D (in-plane) min principal stress
Component.S2SHEAR	2D (in-plane) max shear stress

Component.SAV	Average stress (pressure)
Component.SMAX	Max principal stress
Component.SMID	Middle principal stress
Component.SMIN	Min principal stress
Component.SMS	Max shear stress
Component.SVON	signed von Mises stress
Component.TRI	Triaxiality
Component.YUTF	Yield Utilisation Factor
Component.YUTP	Yield Utilisation Percentage

Constants for Element Stress Tensor Data Components

Name	**Description**
Component.STEN	Stress tensor [SXX, SYY, SZZ, SXY, SYZ, SZX]
Component.SXX	X stress
Component.SXY	XY stress
Component.SYY	Y stress
Component.SYZ	YZ stress
Component.SZX	ZX stress
Component.SZZ	Z stress

Constants for Element Thermal Strain Derived Data Components

Name	**Description**
Component.TEAV	Average thermal strain
Component.TEMAX	Max principal thermal strain
Component.TEMID	Middle principal thermal strain
Component.TEMIN	Min principal thermal strain
Component.TEMS	Max thermal shear strain

Constants for Element Thermal Strain Tensor Data Components

Name	**Description**
Component.TETEN	Thermal strain tensor [EXX, EYY, EZZ, EXY, EYZ, EZX]
Component.TEXX	X Thermal strain
Component.TEXY	XY Thermal shear strain
Component.TEYY	Y Thermal strain
Component.TEYZ	XY Thermal shear strain
Component.TEZX	ZX Thermal shear strain
Component.TEZZ	Z Thermal strain

Constants for Global Energy Data Components

Name	Description
Component.GIE	Internal energy
Component.GKE	Kinetic energy
Component.GTE	Total energy

Constants for Global Mass Data Components

Name	Description
Component.GMASS	Mass

Constants for Global Momentum Data Components

Name	Description
Component.GMM	Momentum magnitude
Component.GMX	X Momentum
Component.GMY	Y Momentum
Component.GMZ	Z Momentum

Constants for Global Velocity Data Components

Name	Description
Component.GVM	Velocity magnitude
Component.GVX	X Velocity
Component.GVY	Y Velocity
Component.GVZ	Z Velocity

Constants for LSDA (binout) Database Cross Section Data Components

Name	Description
Component.XSEC_A	Database X-sect area
Component.XSEC_F	Database X-sect force (vector data)
Component.XSEC_M	Database X-sect moment (vector data)

Constants for LSDA (binout) Retractor Data Components

Name	Description
Component.RT_F	Retractor force
Component.RT_P	Retractor pull-out

Constants for LSDA (binout) SPC Data Components

Name	Description
Component.SPC_F	SPC force (vector data at nodes)
Component.SPC_M	SPC moment (vector data at nodes)

Constants for LSDA (binout) Seatbelt Data Components

Name	Description
Component.SB_F	Seatbelt axial force
Component.SB_L	Seatbelt length

Constants for LSDA (binout) Slipring Data Components

Name	Description
Component.SR_P	Slipring pull-through

Constants for LSDA (binout) Spotweld Data Components

Name	Description
Component.SW_F	Spotweld axial force
Component.SW_FAIL	Spotweld failure
Component.SW_S	Spotweld shear force
Component.SW_TIME	Spotweld failure time
Component.SW_TRSN	Spotweld torsion moment

Constants for LSDA (binout) Spring Data Components

Name	Description
Component.SP_E	Spring elongation
Component.SP_F	Spring axial force
Component.SP_M	Spring torsional moment
Component.SP_R	Spring rotation

Constants for Material Data Components for PARTs and Part-based elems (needs .ZTF file)

Name	Description
Component.DENS	Material density
Component.FSTRN	Failure strain
Component.PRAT	Poisson’s ratio
Component.YMOD	Young’s modulus
Component.YSTRS	Yield stress

Constants for Nastran OP2 Beam Data Components

Name	Description
Component.BENL	Energy loss
Component.BENLD	Energy loss density

Component.BENLP	Energy loss percentage
Component.BKEN	Kinetic energy
Component.BKEND	Kinetic energy density
Component.BKENP	Kinetic energy percentage
Component.BSEN	Strain energy
Component.BSEND	Strain energy density
Component.BSENP	Strain energy percentage

Constants for Nodal Data Components

Name	**Description**
Component.AM	Acceleration magnitude
Component.AV	Acceleration vector [AX, AY, AZ]
Component.AX	X acceleration
Component.AY	Y acceleration
Component.AZ	Z acceleration
Component.BV	Basic (undeformed) vector [BX, BY, BZ]
Component.BX	Basic (undeformed) X coordinate
Component.BY	Basic (undeformed) Y coordinate
Component.BZ	Basic (undeformed) Z coordinate
Component.CV	Current vector [CX, CY, CZ]
Component.CX	Current X coordinate
Component.CY	Current Y coordinate
Component.CZ	Current Z coordinate
Component.DM	Displacement magnitude
Component.DV	Displacement vector [DX, DY, DZ]
Component.DX	X displacement
Component.DY	Y displacement
Component.DZ	Z displacement
Component.RAM	Rotation acceleration magnitude
Component.RAV	Rotation acceleration vector [RAX, RAY, RAZ]
Component.RAX	X rotation acceleration
Component.RAY	Y rotation acceleration
Component.RAZ	Z rotation acceleration
Component.RDM	Rotation displacement magnitude
Component.RDV	Rotation displacement vector [RDX, RDY, RDZ]
Component.RDX	X rotation displacement
Component.RDY	Y rotation displacement
Component.RDZ	Z rotation displacement
Component.RVM	Rotation velocity magnitude

Component.RVV	Rotation velocity vector [RVX, RVY, RVZ]
Component.RVX	X rotation velocity
Component.RVY	Y rotation velocity
Component.RVZ	Z rotation velocity
Component.VM	Velocity magnitude
Component.VV	Velocity vector [VX, VY, VZ]
Component.VX	X velocity
Component.VY	Y velocity
Component.VZ	Z velocity

Constants for Shell and Solid Data Components

Name	**Description**
Component.AREA	Area
Component.DTDT	dTemp / dTime
Component.EDEN	Internal energy density
Component.EMASS	Mass
Component.ENL	Energy loss (Nastran OP2 results only)
Component.ENLD	Energy loss density (Nastran OP2 results only)
Component.ENLP	Energy loss percentage (Nastran OP2 results only)
Component.HGEN	Hourglass energy
Component.KEN	Kinetic energy (Nastran OP2 results only)
Component.KEND	Kinetic energy density (Nastran OP2 results only)
Component.KENP	Kinetic energy percentage (Nastran OP2 results only)
Component.MADD	Added mass
Component.RFX	X force resultant
Component.RFXY	XY force resultant
Component.RFY	Y force resultant
Component.RMX	MX moment resultant
Component.RMXY	MXY moment resultant
Component.RMY	MY moment resultant
Component.RQX	XZ shear force resultant
Component.RQY	YZ shear force resultant
Component.RVOL	Relative volume (solid)
Component.SEN	Strain energy (Nastran OP2 results only)
Component.SEND	Strain energy density (Nastran OP2 results only)
Component.SENP	Strain energy percentage (Nastran OP2 results only)
Component.SHX	Extra shell and thick shell data
Component.SOX	Extra solid data
Component.TBOT	Nodal (shell) bottom surface temperature

Component.TEMP	Nodal temperature
Component.TFM	Temperature magnitude
Component.TFV	Temperature vector [TFX, TFY, TFZ]
Component.TFX	X temperature flux
Component.TFY	Y temperature flux
Component.TFZ	Z temperature flux
Component.THK	Thickness
Component.TMID	Nodal (shell) middle surface temperature
Component.TSTP	Timestep
Component.TTOP	Nodal (shell) top surface temperature
Component.VOL	Volume (solid)

Constants for User defined binary component data type

Name	Description
Component.SCALAR	Scalar data (1 value)
Component.TENSOR	Tensor data (6 values)
Component.VECTOR	Vector data (3 values)

Constants for User defined binary component existing action

Name	Description
Component.RENAME	Renames the name of the component by adding a suffix to make it unique so any existing component of this name (and data) will be left unchanged and the new one will not clash with it
Component.REPLACE	Replaces any existing component, replacing it with this definition. This means that any existing data for the user-defined component of this name is deleted and the component is re-initialised

Constants for User defined binary component location

Name	Description
Component.IN_CORE	held in memory

Constants for User defined binary component type

Name	Description
Component.BEAM	User-defined beam component
Component.NODE	User-defined nodal component
Component.OTHER	User-defined other (LSDA) component
Component.SOLID_SHELL_TSHELL	User-defined solid, shell and thick shell component

Constants for user

Name	Description
Component.UBMS	Beam scalar

Component.UBMV	Beam vector
Component.UNOS	Node scalar
Component.UNOV	Node vector
Component.USSS	Solid and shell scalar
Component.USST	Solid and shell tensor

Component properties

Name	Type	Description
componentType (read only)	constant	The type of component stored in the user defined binary component. Either Component.NODE, Component.BEAM, Component.SOLID_SHELL_TSHELL or Component.OTHER
dataType (read only)	constant	The type of data stored in the user defined binary component. Either Component.SCALAR, Component.TENSOR or Component.VECTOR
dispose (read only)	boolean	If .ubd files for components will be disposed of (deleted) when a model is closed or D3PLOT exits or not. The default is not to delete files.
location (read only)	constant|string	Where the user defined binary component will be written to disk. This can be an absolute or relative pathname. If a relative path is used, this is relative to the Ansys LS-DYNA results files. This is done by giving a pathname beginning with "JOBDIR". For example "JOBDIR/.." refers to the parent directory where the results files are. The default, if location is null, is for .ubd files to be written in the same directory as the Ansys LS-DYNA output files. Alternatively, the data can just be kept in memory and not written to disk by using Component.IN_CORE.
name (read only)	string	The name for the user defined binary component

Detailed Description

The component class allows to to create, modify and delete user defined binary data components in D3PLOT. The class also gives access to various data component constants that are used in the D3PLOT API. They are defined in the Component class so the global namespace is not polluted. See the documentation below for more details.

Constructor

new Component(name*[string]*, component*[constant]*, data*[constant]*, options (optional)*[object]*)

Description

Creates a new user defined binary data component in D3PLOT

Arguments

- **name** (string)

Name for the component

- **component** (constant)

The type of component stored in the user defined binary component. Either Component.NODE, Component.BEAM, Component.SOLID_SHELL_TSHELL or Component.OTHER

- **data** (constant)

The type of data stored in the user defined binary component. Either Component.SCALAR, Component.TENSOR or Component.VECTOR

- **options (optional)** (object)

Object containing extra information. Can contain any of:

Object has the following properties:

Name	Type	Description
dispose	boolean	If .ubd files for components will be disposed of (deleted) when a model is closed or D3PLOT exits or not. The default is not to delete files.
exists	constant	Action to take if a component with this name already exists. Either Component.RENAME or Component.REPLACE (default)
location	constant|string|null	Location to store the .ubd files. See location for details. The default, if location is null, is for .ubd files to be written in the same directory as the Ansys LS-DYNA output files. This default can be changed with the d3plot*ubd_file_location preference.

Returns

Model object

Return type

Model

Example

To create a user defined binary data component in D3PLOT:

```
var ub = new Component("My component", Component.NODE, Component.SCALAR, {
dispose: true, exists: Component.RENAME, location: null });
```

Details of functions

Delete()

Description

Deletes the next user defined binary data component.
Do not use the component object after calling this method

Arguments

No arguments

Returns

Component object

Return type

Component

Example

To delete component c:

```
c.Delete();
```

First() [static]

Description

Returns the first user defined binary component in D3PLOT (or null if there are no components)

Arguments

No arguments

Returns

Component object

Return type

Component

Example

To get the first user defined binary component:

```
var c = Component.First();
```

GetData(item*[Node|Beam|Shell|Solid|Tshell]*, options (optional)*[object]*)

Description

Returns the user defined binary data component for an item

Arguments

- **item** (Node|Beam|Shell|Solid|Tshell)

The Node, Beam, Shell, Solid or Tshell the data should be retrieved for

- **options (optional)** (object)

Object containing extra information. Can contain any of:

Object has the following properties:

Name	Type	Description
ip	integer	The integration point to get the data for for shells and thick shells
op	integer	The on plan integration point to get the data for for fully integrated shells and thick shells. If omitted the first on plan integration point will be used

Returns

The component data

Return type

real|array

Example

To get the data for solid s, user defined component c:

```
var data = c.GetData(s);
```

GetFromID(number*[integer]*) [static]

Description

Returns the user defined binary component in D3PLOT by ID (or null if the component does not exist)

Arguments

- **number** (integer)

number of the component you want the Component object for

Returns

Component object

Return type

Component

Example

To get the Component object for user defined binary component number 1:

```
var c = Component.GetFromID(1);
```

GetFromName(name*[string]*) [static]

Description

Returns the user defined binary component in D3PLOT by name (or null if the component does not exist)

Arguments

- **name** (string)

name of the component you want the Component object for

Returns

Component object

Return type

Component

Example

To get the Component object for user defined binary component named test:

```
var c = Component.GetFromName("test");
```

Last() [static]

Description

Returns the last user defined binary component in D3PLOT (or null if there are no components)

Arguments

No arguments

Returns

Component object

Return type

Component

Example

To get the last user defined binary component:

```
var c = Component.Last();
```

Next()

Description

Returns the next user defined binary data component (or null if there is not one)

Arguments

No arguments

Returns

Component object

Return type

Component

Example

To get the component after component c:

```
c = c.Next();
```

Previous()

Description

Returns the previous user defined binary data component (or null if there is not one)

Arguments

No arguments

Returns

Component object

Return type

Component

Example

To get the component before component c:

```
c = c.Previous();
```

PutData(item*[Node|Beam|Shell|Solid|Tshell]*, data*[real|array]*, options (optional)*[object]*)

Description

Sets the user defined binary data component for an item

Arguments

- **item** (Node|Beam|Shell|Solid|Tshell)

The Node, Beam, Shell, Solid or Tshell the data should be set for

- **data** (real|array)

The data to set. If the component data property is Component.SCALAR this will be a single value. If the component data property is Component.VECTOR this is an array with length 3. If the component data property is Component.TESNOR this is an array with length 6

- **options (optional)** (object)

Object containing extra information. Can contain any of:

Object has the following properties:

Name	Type	Description
ip	integer	The integration point to set the data for for shells and thick shells
op	integer	The on plan integration point to set the data for for fully integrated shells and thick shells

Returns

No return value

Example

To Set the data for solid s, user defined component c to be 1.23:

```
c.PutData(s, 1.23);
```

Total() [static]

Description

Returns the total number of user defined binary components in D3PLOT.

Arguments

No arguments

Returns

Total number of user binary components

Return type

integer

Example

To get total number of components:

```
var total = Component.Total();
```

Constant class

The Constant class defines various constants. More...

The D3PLOT JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Constant constants

Constants for Array indices

Name	Description
Constant.X	Index in a vector/coordinate array containing the X component
Constant.XX	Index in a tensor array containing the XX component
Constant.XY	Index in a tensor array containing the XY component
Constant.Y	Index in a vector/coordinate array containing the Y component
Constant.YY	Index in a tensor array containing the YY component
Constant.YZ	Index in a tensor array containing the YZ component
Constant.Z	Index in a vector/coordinate array containing the Z component
Constant.ZX	Index in a tensor array containing the ZX component
Constant.ZZ	Index in a tensor array containing the ZZ component

Constants for Cut section action

Name	Description
Constant.NORMAL	Positive or negative action Normal
Constant.OMIT	Positive or negative action Omit
Constant.OUTLINE	Positive or negative action Outline (wire drawing)
Constant.TRANSPARENT	Positive or negative action Transparent

Constants for Cut section combination

Name	Description
Constant.INTERSECTION	Intersection mode for multiple cut directions
Constant.UNION	Union mode for multiple cut directions

Constants for Cut section definition

Name	Description
Constant.CONST_X	Constant X definition method

Constant.CONST_Y	Constant Y definition method
Constant.CONST_Z	Constant Z definition method
Constant.LS_DYNA	Ansys LS-DYNA definition method
Constant.N3	3 nodes definition method
Constant.OR_AND_V	Origin and vectors definition method

Constants for Cut section space

Name	Description
Constant.BASIC	Basic (eulerian) space
Constant.DEFORMED	Deformed (lagrangian) space

Constants for Display and overlay mode

Name	Description
Constant.ALL_EDGES	Overlay for all edges
Constant.CURRENT	Current display or overlay mode
Constant.FREE_EDGE	Overlay for free edges
Constant.HIDDEN	Hidden display mode
Constant.NO_OVERLAY	No overlay
Constant.SHADED	Shaded display mode
Constant.WIREFRAME	Wireframe display mode

Constants for Dispose

Name	Description
Constant.DELETE	Delete
Constant.LEAVE	Leave behind (eg don’t delete)

Constants for Frame of Reference

Name	Description
Constant.CYLINDRICAL	Cylindrical coordinate system
Constant.GLOBAL	Global coordinate system
Constant.LOCAL	Element local coordinate system
Constant.MATERIAL	Material axes coordinate system
Constant.USER_DEFINED	User-defined coordinate system

Constants for General

Name	Description
Constant.ALL	All of a category
Constant.GT	Greater than (>)
Constant.GTEQ	Greater than or equals (>=)

Constant.LT	Less than (<)
Constant.LTEQ	Less than or equals (<=)
Constant.MAX	Maximum value
Constant.MIN	Minimum value
Constant.OFF	Switch off
Constant.ON	Switch on

Constants for GetNumberOf

Name	Description
Constant.CUT_SECTION	Number of non-parallel cut plane directions
Constant.FAMILY	Number of families
Constant.INCLUDE	Number of includes
Constant.MODEL	Number of models
Constant.NEIPH	Number of "Extra" Solid variables
Constant.NEIPS	Number of "Extra" Shell variables
Constant.NEIPT	Number of "Extra" Thick Shell variables
Constant.NIP_B	Number of Beam integration points
Constant.NIP_H	Number of Solid integration points
Constant.NIP_S	Number of Shell integration points
Constant.NIP_T	Number of Thick Shell integration points
Constant.N_ON_PLAN	Number of on-plan integration points written
Constant.N_UBMS	Number of user-defined beam scalar components
Constant.N_UBMV	Number of user-defined beam vector components
Constant.N_UNOS	Number of user-defined node scalar components
Constant.N_UNOV	Number of user-defined node vector components
Constant.N_USSS	Number of user-defined solid/shell scalar components
Constant.N_USST	Number of user-defined solid/shell tensor components
Constant.STATE	Number of states
Constant.USER	Number of user-defined components

Constants for Phase Angle Results

Name	Description
Constant.CURRENT_VAL	Current value result
Constant.MAGNITUDE	Magnitude result

Constants for Surface

Name	Description
Constant.AVERAGE	Average across all integration points for beams
Constant.BOTTOM	Bottom shell surface
Constant.MAG_ALL	Value with maximal magintude across all integration points for beams, shells or tshells

Constant.MAX_ALL	Maximum across all integration points for beams, shells or tshells
Constant.MIDDLE	Middle shell surface
Constant.MIN_ALL	Minimum across all integration points for beams, shells or tshells
Constant.TOP	Top shell surface

Detailed Description

The Constant class gives you access to various constants that are used in the D3PLOT API. They are defined in the Constant class so the global namespace is not polluted. See the documentation below for more details.

Contact class

The Contact class gives you access to contacts in D3PLOT. More...

The D3PLOT JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(window*[GraphicsWindow]*, model*[Model]*)
- BlankFlagged(window*[GraphicsWindow]*, model*[Model]*, flag*[Flag]*)
- First(model*[Model]*)
- FlagAll(model*[Model]*, flag*[Flag]*)
- GetAll(model*[Model]*, property (optional)*[string]*)
- GetFlagged(model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(model*[Model]*, label*[integer]*)
- GetFromIndex(model*[Model]*, index*[integer]*)
- GetMultipleData(component*[constant]*, items*[array]*, options (optional)*[object]*)
- Last(model*[Model]*)
- Pick()
- Select(flag*[Flag]*)
- Total(model*[Model]*)
- UnblankAll(window*[GraphicsWindow]*, model*[Model]*)
- UnblankFlagged(window*[GraphicsWindow]*, model*[Model]*, flag*[Flag]*)
- UnflagAll(model*[Model]*, flag*[Flag]*)

Member functions

- Blank(window*[GraphicsWindow]*)
- Blanked(window*[GraphicsWindow]*)
- ClearFlag(flag*[Flag]*)
- Flagged(flag*[Flag]*)
- GetData(component*[constant]*, options (optional)*[object]*)
- GetNode(side*[constant]*, index*[integer]*)
- GetSegment(side*[constant]*, index*[integer]*)
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Unblank(window*[GraphicsWindow]*)

Contact constants

Name	Description
Contact.SURFA	SURFA side of the contact
Contact.SURFB	SURFB side of the contact

Contact properties

Name	Type	Description
aNodes (read only)	integer	Total number of nodes on the SURFA side of the contact
aSegments (read only)	integer	Total number of segments on the SURFA side of the contact

bNodes (read only)	integer	Total number of nodes on the SURFB side of the contact
bSegments (read only)	integer	Total number of segments on the SURFB side of the contact
colour	Colour	The colour of the contact
data (read only)	real|array	Component data for a contact passed as an argument to GetMultipleData. Note that data will only exist for the instance of the contact passed to GetMultipleData. i.e. it is a local property stored on the specific instance. It is not stored in the D3PLOT database
displayMode	integer	The display mode of the contact. Can be Constant.WIREFRAME, Constant.HIDDEN, Constant.SHADED or Constant.CURRENT.
include (read only)	integer	The include file number in the model that the contact is in
index (read only)	integer	The internal index for the contact in D3PLOT (starting at 0)
label (read only)	integer	The Ansys LS-DYNA label for the contact
model (read only)	Model	The Model that the contact is in
name (read only)	string	The name of the contact type
overlayColour	Colour	The overlay colour of the contact
overlayMode	integer	The overlay mode of the contact. Can be Constant.NO_OVERLAY, Constant.FREE_EDGE, Constant.ALL_EDGES or Constant.CURRENT.
title (read only)	string	The title of the contact
transparency	integer	The transparency of the contact between 0 (opaque) and 100 (clear)
type (read only)	constant	The type for the contact (will be Type.CONTACT)

Detailed Description

The Contact class allows you to inspect contacts in a model. See the documentation below for more details.

Details of functions

Blank(window*[GraphicsWindow]*)

Description

Blanks the contact in a graphics window

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to blank the contact in

Returns

No return value

Example

To blank contact c in graphics window g:

```
c.Blank(g);
```

BlankAll(window*[GraphicsWindow]*, model*[Model]*) [static]

Description

Blanks all of the contacts in the model

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to blank the contacts in

- **model** (Model)

Model that all the contacts will be blanked in

Returns

No return value

Example

To blank all of the contacts in model m, in graphics window gw:

```
Contact.BlankAll(gw, m);
```

BlankFlagged(window*[GraphicsWindow]*, model*[Model]*, flag*[Flag]*) [static]

Description

Blanks all of the contacts in the model flagged with a defined flag

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to blank the contacts in

- **model** (Model)

Model that the flagged contacts will be blanked in

- **flag** (Flag)

Flag (see AllocateFlag) set on the contacts to blank

Returns

No return value

Example

To blank all of the contacts flagged with flag f in model m, in graphics window gw:

```
Contact.BlankFlagged(gw, m, f);
```

Blanked(window*[GraphicsWindow]*)

Description

Checks if the contact is blanked in a graphics window or not

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) in which to check if the contact is blanked

Returns

true if blanked, false if not

Return type

boolean

Example

To check if contact c is blanked in graphics window g:

```
if (c.Blanked(g) ) do_something...
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on a contact

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to clear on the contact

Returns

No return value

Example

To clear flag f on contact c:

```
c.ClearFlag();
```

First(model*[Model]*) [static]

Description

Returns the first contact in the model (or null if there are no contacts in the model)

Arguments

- **model** (Model)

Model to get first contact in

Returns

Contact object

Return type

Contact

Example

To get the first contact in model m:

```
var c = Contact.First(m);
```

FlagAll(model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the contacts in the model with a defined flag

Arguments

- **model** (Model)

Model that all the contacts will be flagged in

- **flag** (Flag)

Flag (see AllocateFlag) to set on the contacts

Returns

No return value

Example

To flag all of the contacts with flag f in model m:

```
Contact.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the contact is flagged or not

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to test on the contact

Returns

true if flagged, false if not

Return type

boolean

Example

To check if contact c has flag f set on it:

```
if (c.Flagged(f) ) do_something...
```

GetAll(model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Contact objects or properties for all of the contacts in the model. If the optional property argument is not given then an array of Contact objects is returned. If the property argument is given, that property value for each contact is returned in the array instead of a Contact object

Arguments

- **model** (Model)

Model that all the contacts are in

- **property (optional)** (string)

Name for property to get for all contacts in the model

Returns

Array of Contact objects or properties

Return type

Array

Example

To get all of the contacts in model m:

```
var c = Contact.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'label' for a contact) for each contact in model m:

```
var a = Contact.GetAll(m, 'foo');
```

GetData(component*[constant]*, options (optional)*[object]*)

Description

Returns the value for a data component.
Also see GetMultipleData

Arguments

- **component** (constant)

Component constant to get data for

- **options (optional)** (object)

Object containing options for getting data. Can be any of:

Object has the following properties:

Name	Type	Description
extra	integer	The extra data component number if component Component.SOX for solids, Component.BMX for beams or Component.SHX for shells and thick shells
ip	integer	Integration point number to get the data at (ip >= 1 or one of the constants Constant.TOP, Constant.MIDDLE or Constant.BOTTOM). If the integration point is not defined it will use the integration point defined on the current GUI "data" panel, which defaults to the middle surface for shells, thick shells, and solids, and Mag All for beams, but may vary if changed by an interactive user. If consistent output from a script is required, independent of any prior interactive activity, an explicit integration point or surface should be defined
op	integer	On plane integration point number for shells and thick shells (op >= 1 [default])
referenceFrame	constant	The frame of reference to return values in. Either Constant.GLOBAL (default), Constant.LOCAL, Constant.CYLINDRICAL, Constant.USER_DEFINED or Constant.MATERIAL. This is only necessary for directional components (eg X stress) and then only when something other than the default Constant.GLOBAL coordinate system is to be used
user	integer	The user-defined component number if component Component.UNOS, Component.UNOV, Component.USSS, Component.USST, Component.UBMS or Component.UBMV

Returns

Number if a scalar component, array if a vector or tensor component (or null if the value cannot be calculated because it's not available in the model).
If requesting an invalid component it will throw an error (e.g. Component.AREA of a node).

Return type

real|array

Example

To calculate a component and check it has been calculated (note that in the example, the argument extra is optional):

```
var value = c.GetData(component, {extra: 1});
if (value !== null) do_something...
```

GetFlagged(model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Gets all of the contacts in the model flagged with a defined flag. If the optional property argument is not given then an array of Contact objects is returned. If the property argument is given, that property value for each contact is returned in the array instead of a Contact object

Arguments

- **model** (Model)

Model that the flagged contacts are in

- **flag** (Flag)

Flag (see AllocateFlag) set on the contacts to get

- **property (optional)** (string)

Name for property to get for all flagged contacts in the model

Returns

Array of Contact objects or properties

Return type

Array

Example

To get all of the contacts flagged with flag f in model m:

```
Contact.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'label' for a node) for all contacts flagged with flag f in model m:

```
var a = Contact.GetFlagged(m, f, 'foo');
```

GetFromID(model*[Model]*, label*[integer]*) [static]

Description

Returns the Contact object for contact in model with label (or null if it does not exist)

Arguments

- **model** (Model)

Model to get contact in

- **label** (integer)

The Ansys LS-DYNA label for the contact in the model

Returns

Contact object

Return type

Contact

Example

To get the contact in model m with label 1000:

```
var c = Contact.GetFromID(m, 1000);
```

GetFromIndex(model*[Model]*, index*[integer]*) [static]

Description

Returns the Contact object for contact in model with index (or null if it does not exist)

Arguments

- **model** (Model)

Model to get contact in

- **index** (integer)

The D3PLOT internal index in the model for contact, starting at 0.

Returns

Contact object

Return type

Contact

Example

To get the 51st contact in model m:

```
var c = Contact.GetFromIndex(m, 50);
```

GetMultipleData(component*[constant]*, items*[array]*, options (optional)*[object]*) [static]

Description

Returns the value for a data component for multiple contacts. For each contact a local property called data will be created containing a number if a scalar component, or an array if a vector or tensor component (or null if the value cannot be calculated). The data is also returned as an object.
Also see GetData

Arguments

- **component** (constant)

Component constant to get data for

- **items** (array)

Array of Contact objects to get the data for. All of the contacts must be in the same model.

- **options (optional)** (object)

Object containing options for getting data. Can be any of:

Object has the following properties:

Name	Type	Description
extra	integer	The extra data component number if component Component.SOX for solids, Component.BMX for beams or Component.SHX for shells and thick shells
ip	integer	Integration point number to get the data at (ip >= 1 or one of the constants Constant.TOP, Constant.MIDDLE or Constant.BOTTOM)
op	integer	On plane integration point number for shells and thick shells (op >= 1 [default])
referenceFrame	constant	The frame of reference to return values in. Either Constant.GLOBAL (default), Constant.LOCAL, Constant.CYLINDRICAL, Constant.USER_DEFINED or Constant.MATERIAL. This is only necessary for directional components (eg X stress) and then only when something other than the default Constant.GLOBAL coordinate system is to be used
user	integer	The user-defined component number if component Component.UNOS, Component.UNOV, Component.USSS, Component.USST, Component.UBMS or Component.UBMV

Returns

Object containing the data. A property is created in the object for each contact with the label. The value of the property is a number if a scalar component or an array if a vector or tensor component (or null if the value cannot be calculated)

Return type

object

Example

To calculate a component for contacts in array items and use the data property (note that in the example, the argument extra is optional):

```
Contact.GetMultipleData(component, items, {extra: 1});
for (i=0; i<items.length; i++)
{
    if (items[i].data !== null) do_something...
}
```

To calculate a component for contacts in array items and use the return value (note that in the example, the argument extra is optional):

```
var data = Contact.GetMultipleData(component, items, {extra: 1});
for (d in data)
{
    Message("Label is " + d);
    if (data[d] !== null) do_something...
}
```

GetNode(side*[constant]*, index*[integer]*)

Description

Gets a node for a contact

Arguments

- **side** (constant)

The side of the contact to get the node for. Either Contact.SURFA or Contact.SURFB

- **index** (integer)

index of the node to get.
0 <= index < aNodes for side SURFA
0 <= index < bNodes for side SURFB

Returns

Node object

Return type

Node

Example

To get the 10th node on SURFB side of contact c in D3PLOT

```
var node = c.GetNode(Contact.SURFB, 9);
```

GetSegment(side*[constant]*, index*[integer]*)

Description

Gets a segment for a contact

Arguments

- **side** (constant)

The side of the contact to get the segment for. Either Contact.SURFA or Contact.SURFB

- **index** (integer)

index of the segment to get.
0 <= index < aSegments for side SURFA
0 <= index < bSegments for side SURFB

Returns

Segment object

Return type

Segment

Example

To get the 10th segment on SURFB side of contact c in D3PLOT

```
var segm = c.GetSegment(Contact.SURFB, 9);
```

Last(model*[Model]*) [static]

Description

Returns the last contact in the model (or null if there are no contacts in the model)

Arguments

- **model** (Model)

Model to get last contact in

Returns

Contact object

Return type

Contact

Example

To get the last contact in model m:

```
var c = Contact.Last(m);
```

Next()

Description

Returns the next contact in the model (or null if there is not one)

Arguments

No arguments

Returns

Contact object

Return type

Contact

Example

To get the next contact after contact c:

```
c = c.Next();
```

Pick() [static]

Description

Allows the user to pick a contact from the screen

Arguments

No arguments

Returns

Contact object or null if cancelled

Return type

Contact

Example

To pick a contact:

```
var c = Contact.Pick();
```

Previous()

Description

Returns the previous contact in the model (or null if there is not one)

Arguments

No arguments

Returns

Contact object

Return type

Contact

Example

To get the previous contact before contact c:

```
c = c.Previous();
```

Select(flag*[Flag]*) [static]

Description

Selects contacts using an object menu

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to use when selecting contacts

Returns

The number of contacts selected or null if menu cancelled

Return type

integer

Example

To select contacts, flagging those selected with flag f:

```
var total = Contact.Select(f);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on a contact

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to set on the contact

Returns

No return value

Example

To set flag f on contact c:

```
c.SetFlag(f);
```

Total(model*[Model]*) [static]

Description

Returns the total number of contacts in the model

Arguments

- **model** (Model)

Model to get total in

Returns

The number of contacts

Return type

integer

Example

To get the number of contacts in model m:

```
var total = Contact.Total(m);
```

Unblank(window*[GraphicsWindow]*)

Description

Unblanks the contact in a graphics window

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to unblank the contact in

Returns

No return value

Example

To unblank contact c in graphics window g:

```
c.Unblank(g);
```

UnblankAll(window*[GraphicsWindow]*, model*[Model]*) [static]

Description

Unblanks all of the contacts in the model

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to unblank the contacts in

- **model** (Model)

Model that all the contacts will be unblanked in

Returns

No return value

Example

To unblank all of the contacts in model m, in graphics window gw:

```
Contact.UnblankAll(gw, m);
```

UnblankFlagged(window*[GraphicsWindow]*, model*[Model]*, flag*[Flag]*) [static]

Description

Unblanks all of the contacts in the model flagged with a defined flag

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to unblank the contacts in

- **model** (Model)

Model that the flagged contacts will be unblanked in

- **flag** (Flag)

Flag (see AllocateFlag) set on the contacts to unblank

Returns

No return value

Example

To unblank all of the contacts flagged with flag f in model m, in graphics window gw:

```
Contact.UnblankFlagged(gw, m, f);
```

UnflagAll(model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the contacts in the model

Arguments

- **model** (Model)

Model that the defined flag for all contacts will be unset in

- **flag** (Flag)

Flag (see AllocateFlag) to unset on the contacts

Returns

No return value

Example

To unset flag f on all of the contacts in model m:

```
Contact.UnflagAll(m, f);
```

CutSection class

The CutSection class gives you access to cut sections in D3PLOT. More...

The D3PLOT JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- First(window*[GraphicsWindow object]*)
- FirstFreeDirection(window (optional)*[GraphicsWindow object]*)
- GetAll(window*[GraphicsWindow object]*)
- Last(window*[GraphicsWindow object]*)
- Propagate(options*[object]*)

Member functions

- AddOffsetPlane(offset*[real]*)
- ClearOffsetPlanes()
- GetCutCoords(item*[object]*, options (optional)*[object]*)
- GetForces(options*[object]*)
- GetPlanesForDirection()
- Next()
- Previous()
- SetDefinition(definition*[constant]*, options*[object]*)

CutSection properties

Name	Type	Description
centralPlane (read only)	CutSection object	When multiple cuts are on with either uniform or custom spacing, for any offset plane this property will be the CutSection object for the central cut plane in that direction. Note that if cs is already the central plane or there are no multiple cuts at all, cs.centralPlane will point to the same plane as cs itself.
definition (read only)	constant	The definition method for this cut section. Can be Constant.CONST_X, Constant.CONST_Y, Constant.CONST_Z, Constant.LS_DYNA, Constant.N3 or Constant.OR_AND_V. To change the definition method, see the function CutSection.SetDefinition.
directionID (read only)	integer	Direction index for this cut plane, 1 for D1, 2 for D2 and 3 for D3. When there are multiple cut planes in the same direction, either with uniform spacing or with custom spacing, they will all have the same direction index.
multiCustomCut	logical	true for multiple cuts with custom spacing, false for multiple cuts with uniform spacing. Note that this has an effect only when multiCut is true. If this is changed, CutSection objects for parallel cuts should no longer be referenced except for the local cut origin.
multiCut	logical	Whether or not multiple parallel cuts are on. If this is turned off, CutSection objects for parallel cuts should no longer be referenced except for the local cut origin.
negativeAction	integer	Negative action for the cut plane. Can be Constant.OMIT, Constant.OUTLINE, Constant.NORMAL or Constant.TRANSPARENT.
nneg	integer	Number of parallel planes in negative direction for uniform spacing. If this is changed, CutSection objects for parallel cuts should no longer be referenced except for the local cut origin.

nodes (read only)	array of Node objects	Array of Node objects when cut section defined by 3 nodes or constant X/Y/Z with one node. For multiple parallel cuts this is only defined for the central plane at offset 0.0. To change the nodes, use the function CutSection.SetDefinition.
npos	integer	Number of parallel planes in positive direction for uniform spacing. If this is changed, CutSection objects for parallel cuts should no longer be referenced except for the local cut origin.
offset (read only)	real	Offset of a parallel plane. This will be 0.0 when multiple cuts are off or when this is the central plane, otherwise positive or negative depending on the location of this plane. To change offsets, use the uniformSpacing property on the central plane for uniform spacing or CutSection.ClearOffsetPlanes and CutSection.AddOffsetPlane for custom spacing.
origin (read only)	array of reals	Origin coordinates for this cut plane. For multiple parallel cuts this is only defined for the central plane at offset 0.0. To change the origin, use the function CutSection.SetDefinition.
positiveAction	integer	Positive action for the cut plane. Can be Constant.OMIT, Constant.OUTLINE, Constant.NORMAL or Constant.TRANSPARENT.
thickCut	logical	Whether or not thick cut is on.
thickness	real	Thickness for thick cuts. Note that this affects drawing only when thickCut is set to true.
uniformSpacing	real	Spacing value for parallel cuts with uniform spacing.
visible	logical	whether or not this cut direction is active. For it to be visible, in addition to this visible property also the cutSwitch property on the underlying GraphicsWindow object needs to be on.
window (read only)	GraphicsWindow object	Window where this cut section is defined
xAxis (read only)	array of 3 reals	X axis vector when the cut plane is defined with Origin and Vectors. For multiple parallel cuts this is only defined for the central plane at offset 0.0. To change the X axis vector, use the function CutSection.SetDefinition.
yAxis (read only)	array of 3 reals	Local Y axis vector (normalised). For multiple parallel cuts this is only defined for the central plane at offset 0.0.
zAxis (read only)	array of 3 reals	Local Z axis vector (normalised). For multiple parallel cuts this is only defined for the central plane at offset 0.0.

Detailed Description

The CutSection class allows you to define and modify cut sections in D3PLOT. See the documentation below for more details.

Constructor

new CutSection(definition*[constant]*, options*[object]*)

Description

Creates a new cut section in D3PLOT. If there is already a cut section defined at the specified 'directionID' D1, D2 or D3, this will be replaced by the new one with all properties reset to their defaults except for those specified here.

Arguments

- **definition** (constant)

The definition method for the cut section. Can be Constant.CONST_X, Constant.CONST_Y, Constant.CONST_Z, Constant.LS_DYNA, Constant.N3 or Constant.OR_AND_V.

- **options** (object)

Options to create the cut section.

Object has the following properties:

Name	Type	Description
directionID (optional)	integer	Direction index where the new cut plane definition should be made, 1 for D1, 2 for D2 or 3 for D3. If omitted, the cut plane will be created as direction D1.
nodes	array of Node objects	Node(s) defining the cut section. This is required as array of length 3 when 'definition' is Constant.N3. It can also be specified as array of length 1 in the Constant.CONST_X, Constant.CONST_Y or Constant.CONST_Z case.
normalHead	array of 3 reals	Normal head coordinate for cut plane. This is required when 'definition' is Constant.LS_DYNA.
origin	array of 3 reals	Origin coordinates for cut plane if 'definition' is Constant.CONST_X, Constant.CONST_Y, Constant.CONST_Z without 'nodes' or Constant.OR_AND_V or Constant.LS_DYNA.
window (optional)	GraphicsWindow object	The graphics window where the cut section should be created. If omitted, the cut section will be created in window 1. To define the cut section in all windows, CutSection.Propagate should be used after creating this one.
xAxis	array of 3 reals	X axis vector for cut plane. This is required when 'definition' is Constant.OR_AND_V.
xAxisHead	array of 3 reals	X axis head coordinate for cut plane. This is required when 'definition' is Constant.LS_DYNA.
xyPlane	array of 3 reals	XY plane vector for cut plane. This is required when 'definition' is Constant.OR_AND_V.

Returns

CutSection object

Return type

CutSection

Example

To define a cut section with constant X coordinate 100.0 in GraphicsWindow gw:

```
var cs = new CutSection(Constant.CONST_X, { origin: [100.0, 0.0, 0.0], window: gw });
```

To create a new cut direction with origin 100.0 200.0 300.0, X axis vector 0.0 0.8 0.6 and XY plane vector 1.0 0.0 0.0 in window gw:

```
var dir = CutSection.FirstFreeDirection(gw);
var cs = new CutSection(Constant.OR_AND_V, { origin: [100.0, 200.0, 300.0], xAxis: [0.0, 0.8, 0.6], xyPlane: [1.0, 0.0, 0.0], window: gw, directionID: dir });
```

Details of functions

AddOffsetPlane(offset*[real]*)

Description

Adds another plane with given offset to custom spacing for multiple cuts. This is available only for the central plane. Note that this affects drawing only if multiCut and multiCustomCut are both true.

Arguments

- **offset** (real)

Offset for the new parallel cut plane

Returns

CutSection object

Return type

CutSection

Example

To add a parallel plane to cut section central_cs with offset 10.0 and return a CutSection object for it in offset_cs:

```
var offset_cs = central_cs.AddOffsetPlane(10.0);
```

ClearOffsetPlanes()

Description

Removes all offset planes except 0.0 from custom spacing for multiple cuts. This is available only for the central plane. Note that this affects drawing only if multiCut and multiCustomCut are both true.

Arguments

No arguments

Returns

No return value

Example

To clear all offset planes from a cut section central_cs:

```
central_cs.ClearOffsetPlanes();
```

First(window*[GraphicsWindow object]*) [static]

Description

Returns the first cut section in the window or null if all cut directions D1, D2, D3 are turned off. If for the first active direction there are multiple parallel cuts, this function will return that with the most negative offset. In particular, in the default setup with just one direction D1 and no multiple cuts this function will return 'the' cut section of the graphics window. The main cutting switch has no effect on this function.

Arguments

- **window** (GraphicsWindow object)

Graphics window to return the CutSection object for.

Returns

CutSection object

Return type

CutSection

Example

To get the first cut section in graphics window gw:

```
var cut_section = CutSection.First(gw);
```

FirstFreeDirection(window (optional)*[GraphicsWindow object]*) [static]

Description

Returns the index of the first inactive cut plane direction, i.e. 1 if D1 is not active, 2 if D1 is active but D2 is not, 3 if D1 and D2 are active but D3 is not. If all available directions D1, D2 and D3 are used, this will return 4 although not a valid direction ID.

Arguments

- **window (optional)** (GraphicsWindow object)

Graphics window to return the direction index for. If omitted, the index will be returned based on cut sections in the first graphics window.

Returns

integer

Return type

Number

Example

To get the first free direction index in graphics window gw:

```
var directionID = CutSection.FirstFreeDirection(gw);
```

GetAll(window*[GraphicsWindow object]*) [static]

Description

Returns an array of CutSection objects for all active directions in the window specified or all windows.

Arguments

- **window** (GraphicsWindow object)

Graphics window to return the CutSection objects for. If omitted, CutSection objects for all graphics windows will be returned.

Returns

Array of CutSection objects

Return type

Array

Example

To get an array of all CutSection objects for all graphics windows in the D3PLOT session:

```
var cut_sections = CutSection.GetAll();
```

Note that this would return an array of length 1 with 'the' CutSection object in the most common case of just one graphics window with just one cut plane.

GetCutCoords(item*[object]*, options (optional)*[object]*)

Description

Returns the coordinates where the cut-section cuts through element <type/item>.

Arguments

- **item** (object)

Beam, Shell, Solid or Tshell object

- **options (optional)** (object)

Object has the following properties:

Name	Type	Description
stateID (optional)	integer	State number to be used. If omitted, the current state of the graphics window of the CutSection object will be used.

Returns

2-dimensional array of reals or null if no intersection points. The length of the array returned is the number of intersection points, where each array entry is itself an array of 3 reals for the X, Y and Z coordinates.

Return type

Number

Example

To get the cut coordinates of Shell object s in state 10 with the cut plane represented by the cut section object cs:

```
var coords = cs.GetCutCoords(s, { stateID:10 });
if(coords)
{
    for(var i=0; i<coords.length; i++)
    {
        Message("Intersection point "+(i+1)+": X = "+coords[i][0]+", Y =
"+coords[i][1]+", Z = "+coords[i][2]);
    }
}
else
{
    Message("No intersection points");
}
```

GetForces(options*[object]*)

Description

Returns the forces, moments, centroid and area of the cut section in its window.

The coordinate system of these results depends upon the cut section's space system as follows:

- BASIC space: Forces are moments are always returned in the global axis system, about the geometrical centre of the cut elements at the given state. Therefore the effective origin is likely to change as the model deforms. (This is the method used by LS-DYNA)
- DEFORMED space: Forces and moments are returned in the plane local axis system, about the current section origin. The origin and axes will remain fixed as the model deforms unless one of the "section follows node(s)" options has been used.

WARNING #1: Cut-sections in D3PLOT are a "per window" attribute, cutting all models in a window at the current "frame".

If the optional <stateID> argument is not supplied the forces and moments returned will be at the state of the current "frame" of the window, and while this will normally be the same as the current "state" this is not necessarily the case, since the user may have interpolated results by time

Likewise if the optional <model> argument is not supplied then the model used will be the first in the window (that at index 0 from function GraphicsWindow.GetModelInfo), which may not be the same as the "current model" of the JavaScript interface.

Therefore to avoid ambiguity when extracting cut-section forces and moments it is recommended that:

- The window being used should only contain a single model
- The window should be set up to display all states without interpolation, thus <state id> == <frame id>. (This is the default for windows.)

This "single model in a window" approach is strongly recommended in this context since visual feedback will then match computed values.

WARNING #2: By default computed forces do NOT include blanked elements.

Since cut section display is primarily intended to be used interactively the default behaviour is to omit blanked elements from the forceand moment calculation, since in this way the reported values match what is visible on the screen.

This behaviour is not ideal for batch processing since the user can, by manipulating blanking, change the results which are computed. Therefore the optional argument <includeBlanked> may be used to override this behaviour and to force blanked elements to be considered. If omitted, or set to false, then the default behaviour of omitting blanked elements will continue.

WARNING #3: Cutting a model exactly at mesh lines can result in ill-conditioned force and moment calculation.

It is tempting to define cut planes at nodes since this is easy to do, however this can give rise to ill-conditioning in a rectilinear mesh since the cut may lie exactly on the border between two adjacent elements and therefore won't "know" which one's results to use. Since LS-DYNA elements are constant stress there can be a step change in data values at element borders, and moving the cut plane by a tiny amount can result in a correspondingly large change in cut force and moment values.

It is strongly recommended that cut section definitions used for force and moment extraction should be located away from mesh lines so that they cut elements near their centres, thus avoiding any ambiguity about which elements to use.

WARNING #4: Any element types or parts excluded from the cut section are still included in the force and moment calculation.

Arguments

- **options** (object)

Object has the following properties:

Name	Type	Description
includeBlanked (optional)	logical	false to omit blanked elements (default), true to include them
model (optional)	Model object	Model that exists in the window of this cut section. If omitted the first model in the window will be used.
part (optional)	Part object	If defined, only forces in the specified part are computed.
stateID (optional)	integer	State number to be used. If omitted the state of the window's current frame will be used.

Returns

Object with the following properties:

Name	Type	Description
area	real	Cut section area
centroid	array of reals	Cut section centroid, [Cx, Cy, Cz]
force	array of reals	3 Forces, [Fx, Fy, Fz]
moment	array of reals	3 Moments, [Mxx, Myy, Mzz]

Return type

object

Example

To get the forces of CutSection object cs:

```
if(a = cs.GetForces())
{
    var fx = a.force[Constant.X];
    var myy = a.moment[Constant.Y];
    var area = a.area;
}
```

GetPlanesForDirection()

Description

Returns an array of CutSection objects for all cut planes in the same direction D1, D2 or D3. When there are multiple cuts in either uniform or custom spacing, the array will have one entry for each parallel plane in ascending order of offsets including the central plane itself at offset 0.0. Without multiple cuts the array will just have the plane itself as entry.

Arguments

No arguments

Returns

Array of CutSection objects

Return type

Array

Example

To get an array of all planes in the same direction as cs:

```
var cut_sections = cs.GetPlanesForDirection();
```

Last(window*[GraphicsWindow object]*) [static]

Description

Returns the last cut section in the window or null if all cut directions D1, D2, D3 are turned off. If for the last active direction there are multiple parallel cuts, this function will return that with the most positive offset. The main cutting switch has no effect on this function.

Arguments

- **window** (GraphicsWindow object)

Graphics window to return the CutSection object for.

Returns

CutSection object

Return type

CutSection

Example

To get the last cut section in graphics window gw:

```
var cut_section = CutSection.Last(gw);
```

Next()

Description

Returns the next cut section in the window or null if called for the last one in the graphics window.

Arguments

No arguments

Returns

CutSection object

Return type

CutSection

Example

To loop over all cut sections in graphics window gw:

```
var cs = CutSection.First(gw);
while(cs)
{
    // do something
    cs = cs.Next();
}
```

Previous()

Description

Returns the previous cut section in the window or null if called for the first one in the graphics window.

Arguments

No arguments

Returns

CutSection object

Return type

CutSection

Example

To set CutSection object cs to the previous cut section:

```
cs = cs.Previous();
```

Propagate(options*[object]*) [static]

Description

Propagates the definition of all cut sections to the target window specified.

Arguments

- **options** (object)

Object has the following properties:

Name	Type	Description
source	GraphicsWindow object	Window where the cut section(s) should be propagated from.
target (optional)	GraphicsWindow object	Window where the cut section(s) should be propagated to. If omitted, the cut section(s) will be propagated to all windows in the D3PLOT session.

Returns

No return value

Example

To propagate the cut section(s) from graphics window gw1 to graphics window gw2:

```
CutSection.Propagate({ source: gw1, target: gw2 });
```

SetDefinition(definition*[constant]*, options*[object]*)

Description

Changes the definition method for the cut plane. When there are multiple parallel cuts, this can only be called for the plane at the local cut origin.

Arguments

- **definition** (constant)

The new definition method for the cut section. Can be Constant.CONST_X, Constant.CONST_Y, Constant.CONST_Z, Constant.LS_DYNA, Constant.N3 or Constant.OR_AND_V.

- **options** (object)

Object has the following properties:

Name	Type	Description
nodes	array of Node objects	Node(s) defining the cut section. This is required as array of length 3 when 'definition' is Constant.N3. It can also be specified as array of length 1 in the Constant.CONST_X, Constant.CONST_Y or Constant.CONST_Z case.
normalHead	array of 3 reals	Normal head coordinate for cut plane. This is required when 'definition' is Constant.LS_DYNA.
origin	array of 3 reals	Origin coordinates for cut plane if 'definition' is Constant.CONST_X, Constant.CONST_Y, Constant.CONST_Z without 'nodes' or Constant.OR_AND_V or Constant.LS_DYNA.

xAxis	array of 3 reals	X axis vector for cut plane. This is required when 'definition' is Constant.OR_AND_V.
xAxisHead	array of 3 reals	X axis head coordinate for cut plane. This is required when 'definition' is Constant.LS_DYNA.
xyPlane	array of 3 reals	XY plane vector for cut plane. This is required when 'definition' is Constant.OR_AND_V.

Returns

No return value

Example

To set the definition method for cut section object cs to constant X with coordinate 100.0:

```
cs.SetDefinition({ definition: Constant.CONST_X, origin: [100.0, 0.0, 0.0]});
```

File class

The File class allows you to read and write text files. More...

The D3PLOT JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Copy(source*[string]*, dest*[string]*)
- Delete(filename*[string]*)
- DriveMapFilename(filename*[string]*, format*[constant]*)
- Exists(filename*[string]*)
- FindFiles(directory*[string]*, type (optional)*[constant]*)
- Get(url*[string]*, filename*[string]*, options (optional)*[object]*)
- IsAbsolute(filename*[string]*)
- IsDirectory(filename*[string]*)
- IsFile(filename*[string]*)
- IsReadable(filename*[string]*)
- IsWritable(filename*[string]*)
- Mkdir(directory*[string]*)
- Mktemp()
- Proxy(name*[string]*)
- ProxyPassword(name*[string]*)
- ProxyUsername(username*[string]*)
- ReadCSV(filename*[string]*, delimiter (optional)*[string]*, comment (optional)*[string]*)
- Rename(oldname*[string]*, newname*[string]*)
- Size(filename*[string]*)
- Upload(filename*[string]*, url*[string]*, options (optional)*[object]*)

Member functions

- Close()
- FindLineContaining(contain*[string]*)
- FindLineStarting(start*[string]*)
- Flush()
- ReadAll()
- ReadArrayBuffer(length (optional)*[integer]*)
- ReadChar()
- ReadLine()
- ReadLongLine()
- Seek(offset*[integer]*, origin (optional)*[constant]*)
- Tell()
- Write(string*[Any valid javascript type]*)
- WriteArrayBuffer(buffer*[ArrayBuffer]*, length (optional)*[integer]*)
- Writeln(string*[Any valid javascript type]*)

File constants

Name	Description
File.APPEND	Flag to open file for appending
File.BINARY	Flag to open file in binary mode. This will have no effect on unix/linux but for windows if a file is opened for writing with binary mode \n will not be translated to \r\n (CRLF), it will be written as \n (LF)
File.READ	Flag to open file for reading

File.UTF8	Flag to open file for reading as UTF-8 encoding.
File.WRITE	Flag to open file for writing

Constants for Find types

Name	Description
File.DIRECTORY	Find directories
File.FILE	Find files

Constants for Seek types

Name	Description
File.CURRENT	Seek relative to current file position
File.END	Seek relative to end of the file
File.START	Seek relative to start of the file

File properties

Name	Type	Description
filename (read only)	string	Name of the file
mode (read only)	constant	Mode the file was opened with (File.READ, File.WRITE etc)

Detailed Description

The File class gives you simple functions to read and write text files. The following simple example shows how to read from the file "/data/test/file.txt" and print each line read to the dialogue box:

```
var f, line;
f = new File("/data/test/file.txt", File.READ);
while ( (line = f.ReadLine()) != undefined)
{
    Message(line);
}
f.Close();
```

The following simple example shows how to write the numbers 1 to 10 to the file "/data/test/file.txt":

```
var n, line;
f = new File("/data/test/file.txt", File.WRITE);
for (n=1; n<=10; n++)
{
    f.Writeln(n);
}
f.Close();
```

See the documentation below for more details.

Constructor

new File(filename*[string]*, mode*[constant]*)

Description

Create a new File object for reading and writing text files.

Arguments

- **filename** (string)

Filename of the file you want to read/write. If reading, the file must exist. If writing, the file will be overwritten (if it exists) if mode is File.WRITE, or if mode is File.APPEND it will be appended to if it exists, or created if it does not. When reading a file the filename can also be a URL (uniform resource locator) in which case the file will be read from the remote site. See File.Get() for more details on the format of the URL.

- **mode** (constant)

The mode to open the file with. Can be File.READ, File.WRITE or File.APPEND. For File.WRITE or File.APPEND it can also be ORed with File.BINARY if required. By default text is read and written as ASCII. To read/write text in utf-8 mode can also be ORed with File.UTF8 if required.

Returns

File object

Return type

File

Example

To create a new file object to read file "/data/test/file.txt"

```
var f = new File("/data/test/file.txt", File.READ);
```

Details of functions

Close()

Description

Close a file opened by a File object.

Arguments

No arguments

Returns

No return value

Example

To close File object f.

```
f.Close();
```

Copy(source*[string]*, dest*[string]*) [static]

Description

Copies a file

Arguments

- **source** (string)

Source filename you want to copy.

- **dest** (string)

Destination filename you want to copy source file to.

Returns

true if copy successful, false otherwise.

Return type

Boolean

Example

To copy the file "/data/test/file.key" to "/data/test/file.key_backup"

```
var copied = File.Copy("/data/test/file.key", "/data/test/file.key_backup");
```

Delete(filename*[string]*) [static]

Description

Deletes a file

Arguments

- **filename** (string)

Filename you want to delete.

Returns

true if successful, false if not.

Return type

Boolean

Example

To delete the file "/data/test/file.key"

```
var deleted = File.Delete("/data/test/file.key");
```

DriveMapFilename(filename*[string]*, format*[constant]*) [static]

Description

Changes a filename or directory name to the correct format for a specific operating system using the directory mappings (if present)

Arguments

- **filename** (string)

Filename you want to drive map.

- **format** (constant)

The format for the file/directory name. Can be Include.NATIVE, Include.UNIX or Include.WINDOWS

Returns

string containing drive mapped filename

Return type

String

Example

If D3PLOT has drive S: mapped to "/data" (by using the primer*drive_s, this*drive_s, d3plot*drive_s or oasys*drive_s preference)

```
var mapped = File.DriveMapFilename("/data/test/file.key", Include.WINDOWS);
```

mapped will be "S:\test\file.key".

```
var mapped = File.DriveMapFilename("S:\\test\\file.key", Include.UNIX);
```

mapped will be "/data/test/file.key".

Exists(filename*[string]*) [static]

Description

Check if a file exists. See also File.IsDirectory() and See also File.IsFile().

Arguments

- **filename** (string)

Filename you want to check for existance.

Returns

true/false

Return type

Boolean

Example

To see if the file "/data/test/file.key" exists

```
if (File.Exists("/data/test/file.key")) { do something }
```

FindFiles(directory*[string]*, type (optional)*[constant]*) [static]

Description

Find any files and/or directories in a directory.

Arguments

- **directory** (string)

Directory to look for files/directories in.

- **type (optional)** (constant)

Type of things to find. Can be bitwise OR of File.FILE and File.DIRECTORY. If omitted only files will be returned.

Returns

Array of filenames/directories

Return type

Array

Example

To return the filenames in the directory /data/test:

```
var fileList = File.FindFiles("/data/test")
```

To return the directories in the directory /data/test:

```
var fileList = File.FindFiles("/data/test", File.DIRECTORY)
```

To return the files and directories in the directory /data/test:

```
var fileList = File.FindFiles("/data/test", File.FILE|File.DIRECTORY)
```

FindLineContaining(contain*[string]*)

Description

Reads a line from a file which contains **contain**, opened for reading by a File object. Although this is possible using core JavaScript functions this function should be significantly faster as most of the processing is done by PRIMER in C rather than in the JavaScript interpreter. To enable this function to be as fast as possible a maximum line length of 512 characters is used. If you expect a file to have lines longer than 512 characters then use ReadLongLine which allows lines of any length. If one argument is used then the line must contain that string. If more than one argument is used then lines which contain any of the arguments will be returned

Arguments

- **contain** (string)

String which matching lines must contain

This argument can be repeated if required
Alternatively a single array argument containing the multiple values can be given

Returns

string read from file or `undefined` if end of file

Return type

String

Example

Loop, reading lines from File object f which contain 'example'.

```
var line;
while ( (line = f.FindLineContaining("example") ) != undefined)
{
}
```

FindLineStarting(start*[string]*)

Description

Reads a line from a file which starts with start, opened for reading by a File object. Although this is possible using core JavaScript functions this function should be significantly faster as most of the processing is done by PRIMER in C rather than in the JavaScript interpreter. To enable this function to be as fast as possible a maximum line length of 512 characters is used. If you expect a file to have lines longer than 512 characters then use ReadLongLine which allows lines of any length. If one argument is used then the line must start with that string. If more than one argument is used then lines which start with any of the arguments will be returned

Arguments

- **start** (string)

String which matching lines must start with

This argument can be repeated if required
Alternatively a single array argument containing the multiple values can be given

Returns

string read from file or `undefined` if end of file

Return type

String

Example

Loop, reading lines from File object f which start 'example'.

```
var line;
while ( (line = f.FindLineStarting("example") ) != undefined)
{
}
```

Flush()

Description

Flushes a file opened for writing by a File object.

Arguments

No arguments

Returns

No return value

Example

To flush File object f.

```
f.Flush();
```

Get(url*[string]*, filename*[string]*, options (optional)*[object]*) [static]

Description

Get a file from a remote location. See also File.Proxy(), File.ProxyPassword() and File.ProxyUsername().

Arguments

- **url** (string)

URL (uniform resource locator) of remote file you want to get. Currently http and ftp are supported. For http give the full address including the leading 'http://'. e.g.
'http://www.example.com/file.html'.
For ftp an optional username and password can be given. e.g.
'ftp://ftp.example.com' retrieves the directory listing for the root directory.
'ftp://ftp.example.com/readme.txt' downloads the file readme.txt from the root directory.
'ftp://user:password@ftp.example.com/readme.txt' retrieves the readme.txt file from the user's home directory.

- **filename** (string)

Filename you want to save the file to.

- **options (optional)** (object)

Options for get. If 'username' and 'password' are set then basic authorization using the username and password will be used.

Object has the following properties:

Name	Type	Description

password (optional)	string	Password
response (optional)	boolean	If set to true, then the response code will be returned instead of true/false. This can be used to retieve error messages and codes when the file is not returned successfully.
username (optional)	string	Username

Returns

true if file was successfully got, false otherwise.

Return type

Boolean

Example

To get the file "http://www.example.com/file.html" and save it to C:\temp:

```
File.Get("http://www.example.com/file.html", "C:\temp\file.html");
```

IsAbsolute(filename*[string]*) [static]

Description

Check if a filename is absolute or relative.

Arguments

- **filename** (string)

Filename you want to check.

Returns

true/false

Return type

Boolean

Example

To see if the filename "/data/test" is absolute (which it is!)

```
if (File.IsAbsolute("/data/test")) { do something }
```

IsDirectory(filename*[string]*) [static]

Description

Check if a filename is a directory. See also File.Exists(), File.IsFile(), File.IsReadable() and File.IsWritable().

Arguments

- **filename** (string)

Filename you want to check.

Returns

true/false

Return type

Boolean

Example

To see if the filename "/data/test" is a directory

```
if (File.IsDirectory("/data/test")) { do something }
```

IsFile(filename*[string]*) [static]

Description

Check if a filename is a file. See also File.Exists(), File.IsDirectory(), File.IsReadable() and File.IsWritable().

Arguments

- **filename** (string)

Filename you want to check.

Returns

true/false

Return type

Boolean

Example

To see if the filename "/data/test" is a file

```
if (File.IsFile("/data/test")) { do something }
```

IsReadable(filename*[string]*) [static]

Description

Check if a filename has read permissions. See also File.Exists(), File.IsDirectory() and File.IsWritable().

Arguments

- **filename** (string)

Filename you want to check.

Returns

true/false

Return type

Boolean

Example

To see if the filename "/data/test" is readable

```
if (File.IsReadable("/data/test")) { do something }
```

IsWritable(filename*[string]*) [static]

Description

Check if a filename has write permissions. If *filename* exists and it is a file then it is checked to see if it can be opened with write (File.APPEND permissions). If *filename* exists and it is a directory then the directory is checked for write permission (can files be created in the directory). If *filename* does not exist then it is assumed to be a file and is checked to see if it can be opened for writing (File.WRITE permissions). See also File.Exists(), File.IsDirectory() and File.IsReadable().

Arguments

- **filename** (string)

Filename you want to check.

Returns

true/false

Return type

Boolean

Example

To see if the filename "/data/test" is writable

```
if (File.IsWritable("/data/test")) { do something }
```

Mkdir(directory*[string]*) [static]

Description

Make a directory. If PRIMER preference 'directory_permission' is set e.g.755 then this will apply (same as if set by chmod 755) ignoring any setting of umask. If there is no preference then the users current setting of umask will control permissions (same as system mkdir)

Arguments

- **directory** (string)

The name of the directory you want to create.

Returns

true if successfully created, false if not.

Return type

Boolean

Example

To make the directory "/data/test"

```
var success = File.Mkdir("/data/test");
```

Mktemp() [static]

Description

Make a temporary filename for writing a temporary file.

Arguments

No arguments

Returns

String name of temporary filename that can be used.

Return type

String

Example

To get a temp filename"

```
var filename = File.Mktemp();
```

Proxy(name*[string]*) [static]

Description

Set a proxy for files opened by http, ftp etc. See also File.Get(), File.ProxyPassword() and File.ProxyUsername().

Arguments

- **name** (string)

The name of the proxy.

Returns

No return value

Example

To set the proxy to "http://example.proxy.com" using port 80:

```
File.Proxy("http://example.proxy.com:80");
```

ProxyPassword(name*[string]*) [static]

Description

Set a proxy password for files opened by http, ftp etc. See also File.Get(), File.Proxy() and File.ProxyUsername().

Arguments

- **name** (string)

Password for the proxy server.

Returns

No return value

Example

To set the proxy password to "password":

```
File.ProxyPassword("password");
```

ProxyUsername(username*[string]*) [static]

Description

Set a proxy username for files opened by http, ftp etc. See also File.Get(), File.Proxy() and File.ProxyPassword().

Arguments

- **username** (string)

The username for the proxy.

Returns

No return value

Example

To set the proxy username to "username":

```
File.ProxyUsername("username");
```

ReadAll()

Description

Reads **all** the remaining characters from a file opened for reading by a File object. As this function can read the entire file as a string be careful when reading large files as it will consume large amounts of memory.

Arguments

No arguments

Returns

String. Characters read from file or undefined if end of file

Return type

String

Example

Read all characters from File object f.

```
var c = f.ReadAll();
```

ReadArrayBuffer(length (optional)*[integer]*)

Description

Reads binary data from a file opened for reading by a File object. The data is returned as an ArrayBuffer object. For more details on how to use an ArrayBuffer see the following links:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Typed_arrays
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/ArrayBuffer
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/TypedArray
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/DataView.

Arguments

- **length (optional)** (integer)

Number of bytes to try to read from the file. If omitted all the remaining data from the file will be read.

Returns

ArrayBuffer object or undefined if end of file

Return type

ArrayBuffer

Example

To read data as 32bit unsigned integers from File object f.

```
var ab = f.ReadArrayBuffer();
var u32 = new Uint32Array(ab);
for (var i=0; i<u32.length; i++
{
    var value = u32[i];
}
```

ReadCSV(filename*[string]*, delimiter (optional)*[string]*, comment (optional)*[string]*) [static]

Description

Reads the input CSV file and returns an array of string arrays. If the CSV file has legitimate records the function returns an Array object containing sub-arrays of strings otherwise the function returns NULL. The lengths of all the sub-arrays are the same and equal to maximum number of fields in any of the records. For records in a CSV file having fewer fields, the respective sub-arrays are padded with NULL elements to the maximum array length.

Arguments

- **filename** (string)

Filename you want to read CSV options from.

- **delimiter (optional)** (string)

Delimiter string to be used. Default is a comma (",").

- **comment (optional)** (string)

Comment string to be used. Default is a dollar sign ("$").

Returns

2d array of strings.

Return type

String

Example

To Read CSV file "sample.csv" and print all records to a Window.

```
var csv_file_path = "C:\\sample.csv";
var records = "";
if(!File.Exists(csv_file_path))
{
    Window.Information("CSV file %s not present", csv_file_path);
    Exit();
}
var csv_array = File.ReadCSV(csv_file_path);
if(csv_array != null)
{
    for(var i = 0; i < csv_array.length; i++)
    {
        var record_array = csv_array[i];
        for(var j = 0; j < record_array.length; j++)
        {
            if(record_array[j] != null)
                records = records + record_array[j] + " , ";
        }
        records = records + "\n";
    }
}
Options.max_window_lines = csv_array.length;
Window.Information("File.ReadCSV Ouptut", records);
```

To Read CSV file "sample.csv" with delimiter string "::" and comment string "##".

```
var csv_array = File.ReadCSV(csv_file_path, "::", "##");
```

ReadChar()

Description

Reads a single character from a file opened for reading by a File object.

Arguments

No arguments

Returns

character read from file or

`undefined`

if end of file

Return type

String

Example

Loop, reading characters from File object f.

```
var c;
while ( (c = f.ReadChar()) != undefined) { ... }
```

ReadLine()

Description

Reads a line from a file opened for reading by a File object. To enable this function to be as fast as possible a maximum line length of 512 characters is used. If you expect a file to have lines longer than 512 characters then use ReadLongLine which allows lines of any length.

Arguments

No arguments

Returns

string read from file or

`undefined`

if end of file

Return type

String

Example

Loop, reading lines from File object f.

```
var line;
while ( (line = f.ReadLine()) != undefined) { ... }
```

ReadLongLine()

Description

Reads a line from a file opened for reading by a File object. The line can be any length. If your file has lines shorter than 512 characters then you may want to use ReadLine instead which is faster.

Arguments

No arguments

Returns

string read from file or

`undefined`

if end of file

Return type

String

Example

Loop, reading lines from File object f.

```
var line;
while ( (line = f.ReadLongLine()) != undefined) { ... }
```

Rename(oldname*[string]*, newname*[string]*) [static]

Description

Rename an existing file to have a different name.

Arguments

- **oldname** (string)

Existing filename you want to rename

- **newname** (string)

New filename you want to rename to

Returns

true if successful, false if not.

Return type

Boolean

Example

To rename the file "/data/test/file.key" to "/data/test/new_file.key"

```
var size = File.Rename("/data/test/file.key", "/data/test/new_file.key");
```

Seek(offset*[integer]*, origin (optional)*[constant]*)

Description

Set the current position for reading or writing in a File object.

Arguments

- **offset** (integer)

Offset to seek to in the file

- **origin (optional)** (constant)

Origin for offset. Must be one of File.START, File.END or File.CURRENT. If omitted File.START will be used.

Returns

no return value

Example

To seek to the end of File f:

```
f.Seek(0, File.END);
```

To seek to the beginning of File f:

```
f.Seek(0, File.START);
```

To move forward 10 characters in File f:

```
f.Seek(10, File.CURRENT);
```

Size(filename*[string]*) [static]

Description

Return the size of a file in bytes

Arguments

- **filename** (string)

Filename you want the size of.

Returns

size in bytes

Return type

Number

Example

To get the size of the file "/data/test/file.key"

```
var size = File.Size("/data/test/file.key");
```

Tell()

Description

Return the current file position for a File object. Note that on Windows when reading files if the file is not opened with File.BINARY this may not return the correct file position for files with unix line endings.

Arguments

No arguments

Returns

integer

Return type

Number

Example

To get the current file position for File f:

```
var pos = f.Tell();
```

Upload(filename*[string]*, url*[string]*, options (optional)*[object]*) [static]

Description

Uploads a file to a remote location. See also File.Proxy(), File.ProxyPassword() and File.ProxyUsername().

Arguments

- **filename** (string)

Filename you want to upload.

- **url** (string)

URL (uniform resource locator) of the remote location you want to upload the file to. Currently only http is supported. Give the full address including the leading 'http://'. e.g. 'http://www.example.com/file.html'.

- **options (optional)** (object)

Options for upload. If both of these are set then basic authorization using the username and password will be used.

Object has the following properties:

Name	Type	Description
password (optional)	string	Password
username (optional)	string	Username

Returns

true if file was successfully uploaded, false otherwise.

Return type

Boolean

Example

To upload the file "C:\temp\file.txt" to "http://www.example.com/file.txt":

```
File.Upload("C:/temp/file.txt", "http://www.example.com/file.txt");
```

Write(string*[Any valid javascript type]*)

Description

Write a string to a file opened for writing by a File object. **Note that a carriage return is not added**.

Arguments

- **string** (Any valid javascript type)

The string/item that you want to write

Returns

No return value

Example

To write string "Hello, world!" to File object f

```
f.Write("Hello, world!\n");
```

To write the title of model m to File object f

```
f.Write("The title of model 2 is " + m.title + "\n");
```

WriteArrayBuffer(buffer*[ArrayBuffer]*, length (optional)*[integer]*)

Description

Writes binary data to a file opened for writing by a File object. The data to write is an ArrayBuffer object. For more details on how to use an ArrayBuffer see the following links:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Typed_arrays
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/ArrayBuffer
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/TypedArray
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/DataView.

Arguments

- **buffer** (ArrayBuffer)

ArrayBuffer to write to file

- **length (optional)** (integer)

Number of bytes to write to the file. If omitted all the data in the ArrayBuffer will be written (buffer.byteLength bytes)

Returns

No return value

Example

To write ArrayBuffer ab to File object f.

```
f.WriteArrayBuffer(ab);
```

Writeln(string*[Any valid javascript type]*)

Description

Write a string to a file opened for writing by a File object **adding a carriage return**.

Arguments

- **string** (Any valid javascript type)

The string/item that you want to write

Returns

No return value

Example

To write string "Hello, world!" to File object f automatically adding a carriage return

```
f.Writeln("Hello, world!");
```

To write the title of model m to File object f automatically adding a carriage return

```
f.Writeln("The title of model 2 is " + m.title);
```

GraphicsWindow class

The GraphicsWindow class gives you access to graphics windows in D3PLOT. More...

The D3PLOT JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- First()
- GetFromID(graphics window number*[integer]*)
- Last()
- StateChangeFunction(name*[function]*)
- Total()

Member functions

- AddModel(model*[Model object]*)
- Delete()
- GetModelInfo(index*[integer]*)
- GetTargetEye()
- Next()
- Previous()
- Redraw()
- RemoveModel(model*[Model object]*)
- SetModelInfo(index*[integer]*, info*[object]*)
- SetTargetEye(info*[object]*)

GraphicsWindow constants

Constants for model offset

Name	Description
GraphicsWindow.NO_OFFSET	Model has no offset in the graphics window
GraphicsWindow.OFFSET_MODEL_SPACE	Model is offset in model space coordinates
GraphicsWindow.OFFSET_SCREEN_SPACE	Model is offset in screen space coordinates

Constants for up vector

Name	Description
GraphicsWindow.UP_AUTOMATIC	Automatically calculate up vector for target and eye
GraphicsWindow.UP_X	Set up vector for target and eye to global X
GraphicsWindow.UP_Y	Set up vector for target and eye to global Y
GraphicsWindow.UP_Z	Set up vector for target and eye to global Z

GraphicsWindow properties

Name	Type	Description
active	boolean	Whether the graphics window is active or not (equivalent to turning the window off/on in the GUI)
cutCombination	integer	How positive and negative actions for multiple cut section directions are combined. Can be Constant.UNION or Constant.INTERSECTION. Note that this only has an effect with at least 2 cut plane directions.
cutFollowN	logical	true if cut sections in this window follow nodes in this window, false otherwise. This only has an effect for definitions by constant X/Y/Z with node or 3 nodes.
cutSpace	integer	Cut space system. Can be Constant.BASIC or Constant.DEFORMED (default).
cutSwitch	logical	true if the cutting switch is on in this window, false otherwise
models (read only)	integer	The total number of models in this graphics window
number (read only)	integer	The graphics window number
state	integer	The current state displayed in this graphics window. Also see the Model state.
states (read only)	integer	The highest state number from all models in this graphics window

Detailed Description

The GraphicsWindow class allows you to do various operations on graphics windows in D3PLOT. There are various methods and properties available that allow you do create, delete and modify thems. See the documentation below for more details.

Constructor

new GraphicsWindow(model*[Model object]*)

Description

Creates a new graphics window in D3PLOT

Arguments

- **model** (Model object)

The model to open in this graphics window

This argument can be repeated if required
Alternatively a single array argument containing the multiple values can be given

Returns

GraphicsWindow object

Return type

GraphicsWindow

Example

To create a graphics window containing Model m in D3PLOT

```
var gw = new GraphicsWindow(m);
```

Details of functions

AddModel(model*[Model object]*)

Description

Adds a model to a graphics window

Arguments

- **model** (Model object)

The model to add to the graphics window

Returns

No return value

Example

To add model m to graphics window gw in D3PLOT

```
gw.AddModel(m);
```

Delete()

Description

Deletes a graphics window in D3PLOT
Do not use the GraphicsWindow object after calling this method.

Arguments

No arguments

Returns

No return value

Example

To delete graphics window gw in D3PLOT

```
gw.Delete();
```

First() [static]

Description

Returns the GraphicsWindow object for the first graphics window in D3PLOT (or null if there are no graphics windows)

Arguments

No arguments

Returns

GraphicsWindow object

Return type

GraphicsWindow

Example

To get the GraphicsWindow object for the first graphics window:

```
var gw = GraphicsWindow.First();
```

GetFromID(graphics window number*[integer]*) [static]

Description

Returns the GraphicsWindow object for a graphics window ID (or null if graphics window does not exist)

Arguments

- **graphics window number** (integer)

number of the graphics window you want the GraphicsWindow object for

Returns

GraphicsWindow object

Return type

GraphicsWindow

Example

To get the GraphicsWindow object for graphics window number 1

```
var gw = GraphicsWindow.GetFromID(1);
```

GetModelInfo(index*[integer]*)

Description

Gets the information for a model in a graphics window

Arguments

- **index** (integer)

index of the model in the graphics window you want information for (0 <= index < models)

Returns

Object with the following properties:

Name	Type	Description
colour	Colour	The colour for the model
mode	constant	How the model is model is displayed in the graphics window. One of View.WIRE, View.HIDDEN, View.SHADED or View.CURRENT
model	Model object	The model at the given index
offsetMode	constant	How the model is offset in the graphics window. One of GraphicsWindow.NO_OFFSET, GraphicsWindow.OFFSET_MODEL_SPACE or GraphicsWindow.OFFSET_SCREEN_SPACE
state	integer	The current state number for the model
visible	boolean	Whether the model is visible in the graphics window or not
xOffset	real	The X offset for the model

yOffset	real	The Y offset for the model
zOffset	real	The Z offset for the model

Return type

object

Example

To get the information for the second model in graphics window gw:

```
var info = gw.GetModelInfo(1);
```

GetTargetEye()

Description

Get the current target and eye settings

Arguments

No arguments

Returns

Object with the following properties:

Name	Type	Description
eye	array	Array containing the coordinates of the eye position
perspective	boolean	Whether the perspective switch is active
scale	real	Scale factor
target	array	Array containing the coordinates of the target position
up	constant/array	GraphicsWindow.UP_AUTOMATIC, GraphicsWindow.UP_X, GraphicsWindow.UP_Y, GraphicsWindow.UP_Z or an array containing the up vector if user defined

Return type

object

Example

To get the current target and eye information for graphics window gw:

```
var info = gw.GetTargetEye();
```

Last() [static]

Description

Returns the GraphicsWindow object for the last graphics window in D3PLOT (or null if there are no graphics windows)

Arguments

No arguments

Returns
GraphicsWindow object

Return type
GraphicsWindow

Example
To get the GraphicsWindow object for the last graphics window:

```
var gw = GraphicsWindow.Last();
```

Next()

Description
Returns the next graphics window (or null if there is not one)

Arguments
No arguments

Returns
GraphicsWindow object

Return type
GraphicsWindow

Example
To get the graphics window after graphics window gw:

```
gw = gw.Next();
```

Previous()

Description
Returns the previous graphics window (or null if there is not one)

Arguments
No arguments

Returns
GraphicsWindow object

Return type
GraphicsWindow

Example
To get the graphics window before graphics window gw:

```
gw = gw.Previous();
```

Redraw()

Description

Redraws the graphics window

Arguments

No arguments

Returns

No return value

Example

To dedraw graphics window gw:

```
gw.Redraw();
```

RemoveModel(model*[Model object]*)

Description

Removes a model from a graphics window

Arguments

- **model** (Model object)

The model to remove from the graphics window

Returns

No return value

Example

To remove model m from graphics window gw in D3PLOT

```
gw.RemoveModel(m);
```

SetModelInfo(index*[integer]*, info*[object]*)

Description

Sets the information for a model in a graphics window

Arguments

- **index** (integer)

index of the model in the graphics window you want to set information for (0 <= index < models)

- **info** (object)

Object containing the information to set. Can be any of:

Object has the following properties:

Name	Type	Description
colour	Colour	The colour for the model
mode	constant	How the model is model is displayed in the graphics window. One of View.WIRE, View.HIDDEN, View.SHADED or View.CURRENT

offsetMode	constant	How the model is offset in the graphics window. One of GraphicsWindow.NO_OFFSET, GraphicsWindow.OFFSET_MODEL_SPACE or GraphicsWindow.OFFSET_SCREEN_SPACE
visible	boolean	Whether the model is visible in the graphics window or not
xOffset	real	The X offset for the model
yOffset	real	The Y offset for the model
zOffset	real	The Z offset for the model

Returns

No return value

Example

To set the second model in graphics window gw to have an offset of (100, 100, 0) in model space:

```
gw.SetModelInfo(1, { offsetMode: GraphicsWindow.OFFSET_MODEL_SPACE, xOffset:
100, yOffset:100, zOffset: 0 } );
```

SetTargetEye(info*[object]*)

Description

Set the current target and eye settings

Arguments

- **info** (object)

Object containing the target and eye properties

Object has the following properties:

Name	Type	Description
eye	array	Array containing the coordinates of the eye position
perspective	boolean	Whether the perspective switch is active. If omitted the default is true
scale	real	Scale factor. If omitted, the scaling is automatically calculated (in the same way as if running D3PLOT interactively)
target	array	Array containing the coordinates of the target position
up	constant/array	GraphicsWindow.UP_AUTOMATIC, GraphicsWindow.UP_X, GraphicsWindow.UP_Y, GraphicsWindow.UP_Z or an array containing the up vector if user defined. If omitted the default is GraphicsWindow.UP_AUTOMATIC

Returns

No return value

Example

To set the eye position to (-2365.1, -373.2, 1081.1) looking at target (-2075.9, -367.7, 905.2) with the up vector in global Z, automatically turning on perspective and calculating the scale factor, in graphics window gw:

```
gw.SetTargetEye( { eye: [-2365.1, -373.2, 1081.1], target: [-2075.9, -367.7,
905.2], up: GraphicsWindow.UP_Z } );
```

To set the eye position to (-2365.1, -373.2, 1081.1) looking at target (-2075.9, -367.7, 905.2) with the up vector in global Z, turning off perspective and setting the scale factor to 10, in graphics window gw:

```
gw.SetTargetEye( { eye: [-2365.1, -373.2, 1081.1], target: [-2075.9, -367.7,
905.2], up: GraphicsWindow.UP_Z, perspective: false, scale: 10 } );
```

StateChangeFunction(name*[function]*) [static]

Description

Defines a state change callback function

Arguments

- **name** (function)

The name of the function (or null to remove a function). When defined, this function will be called every time that the state changes in a graphics window. In the definition of this callback function the keyword 'this' will be a GraphicsWindow object pointing to the window where the state has changed. When the state changes simultaneously for multiple graphics windows, there will be multiple calls to this callback function, one for each graphics window where the state is changed.

Returns

No return value

Example

To define a state change callback:

```
GraphicsWindow.StateChangeFunction(my_callback);
function my_callback()
{
    Message("The state has changed in graphics window " + this.number + ".");
    for(var i = 0; i < this.models; i++)
    {
        var info = this.GetModelInfo(i);
        Message("Model " + info.model.number + " is at state " + info.state + "
in this window.");
    }
    // do something
}
```

Total() [static]

Description

Returns the total number of graphics windows in use in D3PLOT

Arguments

No arguments

Returns

Total number of graphics windows

Return type

integer

Example

To get total number of graphics windows:

```
var total = GraphicsWindow.Total();
```

Group class

The Group class gives you access to groups in D3PLOT. More...

The D3PLOT JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- First(model*[Model]*)
- GetFromID(model*[Model]*, label*[integer]*)
- Last(model*[Model]*)
- Total(model*[Model]*)

Member functions

- AddFlagged(flag*[Flag]*)
- Empty()
- FlagContents(flag*[Flag]*)
- Next()
- Previous()

Group properties

Name	Type	Description
label (read only)	integer	The group label
model (read only)	Model	The Model that the group is in
title	string	The group title
type (read only)	constant	The type for the group (will be Type.GROUP)

Detailed Description

The Group class allows you to inspect groups in a model. See the documentation below for more details.

Constructor

new Group(model*[Model object]*)

Description

Creates a new group in D3PLOT

Arguments

- **model** (Model object)

The model to create the group in

Returns

Group object

Return type

Group

Example

To create a group for Model m in D3PLOT

```
var g = new Group(m);
```

Details of functions

AddFlagged(flag*[Flag]*)

Description

Adds flagged items to the contents of the group

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) set on items to add to the group

Returns

No return value

Example

To add items flagged with flag f to group g:

```
g.AddFlagged(f);
```

Empty()

Description

Empties the group (removes everything from the group)

Arguments

No arguments

Returns

No return value

Example

To empty group g:

```
g.Empty();
```

First(model*[Model]*) [static]

Description

Returns the first group in the model (or null if there are no groups)

Arguments

- **model** (Model)

Model to get first group in

Returns

Group object

Return type

Group

Example

To get the first group in model m:

```
var g = Group.First(m);
```

FlagContents(flag*[Flag]*)

Description

Flags the contents of the group

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to set for the group contents

Returns

No return value

Example

To flag the contents of group g with flag f:

```
g.FlagContents(f);
```

GetFromID(model*[Model]*, label*[integer]*) [static]

Description

Returns the Group object for group in model with label (or null if it does not exist)

Arguments

- **model** (Model)

Model to get group in

- **label** (integer)

The label for the group in the model

Returns

Group object

Return type

Group

Example

To get the group in model m with label 2:

```
var g = Group.GetFromID(m, 2);
```

Last(model[*Model*]) [static]

Description

Returns the last group in the model (or null if there are no groups)

Arguments

- **model** (Model)

Model to get last group in

Returns

Group object

Return type

Group

Example

To get the last group in model m:

```
var g = Group.Last(m);
```

Next()

Description

Returns the next group in the model (or null if there is not one)

Arguments

No arguments

Returns

Group object

Return type

Group

Example

To get the next group after group g:

```
g = g.Next();
```

Previous()

Description

Returns the previous group in the model (or null if there is not one)

Arguments

No arguments

Returns

Group object

Return type

Group

Example

To get the previous group before group g:

```
g = g.Previous();
```

Total(model[*Model*]) [static]

Description

Returns the total number of groups in a model

Arguments

- **model** (Model)

Model to get group in

Returns

The number of groups

Return type

integer

Example

To get the number of groups in model m:

```
var total = Group.Total(m);
```

Image class

The Image class allows you to write image and movie files from D3PLOT. More...

The D3PLOT JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Write3D(name*[string]*, options (optional)*[object]*)
- WriteImage(name*[string]*, options (optional)*[object]*)
- WriteMovie(name*[string]*, options (optional)*[object]*)

Image constants

Constants for 3D (D3PLOT viewer types

Name	Description
Image.GLBU	Uncompressed GLB animation file

Constants for 3D (D3PLOT viewer) types

Name	Description
Image.GLB	GLB animation file

Constants for image types

Name	Description
Image.BMP	24-bit uncompressed BMP file
Image.BMP8	8-bit uncompressed BMP file
Image.BMP8C	8-bit compressed BMP file
Image.GIF	8-bit GIF file
Image.JPEG	JPEG file
Image.PNG	24-bit PNG file
Image.PNG8	8-bit PNG file
Image.PPM	PPM file

Constants for movie types

Name	Description
Image.AGIF	8-bit animated GIF file
Image.AVI	AVI movie file
Image.MP4	MP4 movie file

Constants for resolution

Name	Description
Image.SCREEN	Image will be created at screen resolution.
Image.X2	Image will be created at 2x screen resolution.
Image.X4	Image will be created at 4x screen resolution.

Constants for state selection

Name	Description
Image.ALL_STATES	Use all states

Detailed Description

The Image class gives you methods to be able to write images, animations and 3D models. See the documentation below for more details.

Details of functions

Write3D(name*[string]*, options (optional)*[object]*) [static]

Description

Writes a 3D (GLB) file

Arguments

- **name** (string)

Filename for the movie

- **options (optional)** (object)

Object containing options for writing movie. Can be any of:

Object has the following properties:

Name	Type	Description
anonymise (optional)	boolean	Anonymise part tree (default = false)
format (optional)	constant	The format for the 3D file. Either Image.GLB (default) or Image.GLBU
frameRate (optional)	integer	The frame rate if multiple states are written (default = 25)
graphicsWindow (optional)	GraphicsWindow	An individual GraphicsWindow to write 3D data for. If not given (default) all the windows on the current page will be written
state (optional)	integer	The state number to write or Image.ALL_STATES (default)

Returns

No return value

Example

To write a 3D file "example.glb" for graphics window gw:

```
Image.Write3D("example.glb", { 'graphicsWindow': gw } );
```

WriteImage(name*[string]*, options (optional)*[object]*) [static]

Description

Writes a static image file

Arguments

- **name** (string)

Filename for the image

- **options (optional)** (object)

Object containing options for writing image. Can be any of:

Object has the following properties:

Name	Type	Description
dither (optional)	boolean	Whether to dither 8 bit images or not (default is false)
format (optional)	constant	The format for the image. One of Image.BMP8C, Image.BMP8, Image.BMP, Image.JPEG, Image.PPM, Image.PNG (default), Image.PNG8 or Image.GIF
graphicsWindow (optional)	GraphicsWindow	An individual GraphicsWindow to write image for. If not given (default) all the windows on the current page will be written
optimise (optional)	boolean	Whether to optimise the colour palette for 8 bit images or not (default is true)
quality (optional)	integer	The quality for JPEG images (1 - 100) (default = 90)
resolution (optional)	constant	The resolution for the image. One of Image.SCREEN (default), Image.X2 or Image.X4

Returns

No return value

Example

To write an image "example.png"

```
Image.WriteImage("example.png");
```

WriteMovie(name*[string]*, options (optional)*[object]*) [static]

Description

Writes a movie file

Arguments

- **name** (string)

Filename for the movie

- **options (optional)** (object)

Object containing options for writing movie. Can be any of:

Object has the following properties:

Name	Type	Description
aviFormat (optional)	constant	The AVI format for the movie if writing an AVI file. One of Image.JPEG (default), Image.BMP8C, Image.BMP8 or Image.BMP
dither (optional)	boolean	Whether to dither 8 bit images or not (default is false)

format (optional)	constant	The format for the movie. One of Image.MP4 (default), Image.AVI or Image.AGIF
frameRate (optional)	integer	The frame rate (default = 25)
graphicsWindow (optional)	GraphicsWindow	An individual GraphicsWindow to write movie for. If not given (default) all the windows on the current page will be written
quality (optional)	integer	The quality for AVI MJPEG images (1 - 100) (default = 90)
repeat (optional)	integer	The number of repeats for the movie. 0 is infinite (default = 0)
targetFileSize (optional)	real	The target file size for the movie file in megabytes. This only applies when 'format' is Image.MP4. If this is positive, it supersedes any value for 'quality'. This value is only a target and not a certainly achieved exact size. If the target file size is 0.0, it has the same effect as being undefined.

Returns

No return value

Example

To write a movie "example.mp4":

```
Image.WriteMovie("example.mp4");
```

Include class

The Include class gives you access to include files in D3PLOT. More...

The D3PLOT JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- First(model*[Model]*)
- GetFromID(model*[Model]*, number*[integer]*)
- Last(model*[Model]*)
- Total(model*[Model]*)

Member functions

- Next()
- Previous()

Include constants

Constants for Directory separators

Name	Description
Include.NATIVE	Use directory separators native to this machine when writing directory names.
Include.UNIX	Use unix directory separators when writing directory names.
Include.WINDOWS	Use windows directory separators when writing directory names.

Include properties

Name	Type	Description
label (read only)	integer	The label for the include file
name (read only)	string	The name for the include file
parent (read only)	integer	The label for the include file parent (0 if main file)

Detailed Description

The Include class allows you to inspect include files that are used in a model. Note that for D3PLOT to be able to get include file data there must be a ztf file. See the documentation below for more details.

Details of functions

First(model*[Model]*) [static]

Description

Returns the first include file in the model (or null if there are no include files in the model)

Arguments

- **model** (Model)

Model to get first include file in

Returns

Include object

Return type

Include

Example

To get the first include file in model m:

```
var i = Include.First(m);
```

GetFromID(model*[Model]*, number*[integer]*) [static]

Description

Returns the include file in the model with number (or null if it does not exist)

Arguments

- **model** (Model)

Model to get include file in

- **number** (integer)

The number for the include file in the model. Note that include file numbers start at 1. 0 is the main file.

Returns

Include object

Return type

Include

Example

To get include file number 5 in model m:

```
var i = Include.GetFromID(m, 5);
```

Last(model*[Model]*) [static]

Description

Returns the last include file in the model (or null if there are no include files in the model)

Arguments

- **model** (Model)

Model to get last include file in

Returns

Include object

Return type

Include

Example

To get the last include file in model m:

```
var i = Include.Last(m);
```

Next()

Description

Returns the next include file in the model (or null if there is not one)

Arguments

No arguments

Returns

Include object

Return type

Include

Example

To get the next include file after include i:

```
var i = i.Next();
```

Previous()

Description

Returns the previous include file in the model (or null if there is not one)

Arguments

No arguments

Returns

Include object

Return type

Include

Example

To get the previous include file before include i:

```
var i = i.Previous();
```

Total(model[*Model*]) [static]

Description

Returns the total number of include files in the model

Arguments

- **model** (Model)

Model to get total in

Returns

Number of includes

Return type

int

Example

To get the number of include files in model m:

```
var t = Include.Total(m);
```

Link class

The Link class allows you to link to other Oasys LS-DYNA Environment programs on the computer. More...

The D3PLOT JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- ConnectPending()

Member functions

- Send(data*[ArrayBuffer or string]*)
- Terminate()

Link constants

Constants for programs

Name	Description
Link.D3PLOT	Link to D3PLOT
Link.PRIMER	Link to PRIMER
Link.THIS	Link to T-HIS

Constants for status

Name	Description
Link.CONNECTED	Link is connected to the remote script
Link.CREATED	Link has been created but not connected to the remote script
Link.DISCONNECTED	Link has been disconnected as the remote script has finished/exited
Link.TERMINATED	Link has been terminated as the remote program has closed/crashed

Link properties

Name	Type	Description
onDisconnect	function	Function to call when when the link is disconnected, either because the remote program has terminated, or the remote script has finished. The status property can be used to distinguish these cases. The function has no arguments. The Link object is accessible in the function using the 'this' keyword. To unset the function, set the property to null.

onReceipt	function	Function to call to acknowledge receipt of the data in the remote program (data sent from a Send command). The function takes a single argument which is the UUID returned from the original Send command. Note that this function will only be called if the link is asynchronous. In the synchronous case the local program will wait for the remote program to process the Send command so this is unnecessary. To unset the function, set the property to null.
onReceive	function	Function to call when data is received from a Send command in the remote program. The function takes a single argument which contains the data passed by Send. To unset the function, set the property to null.
size (read only)	integer	Size of shared memory buffer in KB
status (read only)	constant	The status of the connection. Can be Link.CREATED, Link.CONNECTED, Link.DISCONNECTED or Link.TERMINATED
synchronous	boolean	whether link Send commands are synchronous or asynchronous (default)
timeout	integer	Timeout for Send commands in msecs. If equal to 0 (default) then a default timeout of 5s is used

Detailed Description

The Link class gives you simple functions for a script to pass information to and from scripts running on the other Oasys LS-DYNA Environment programs on the same computer.
It is the JavaScript equivalent of the user pressing the 'Post' button in PRIMER to start D3PLOT or T/HIS, or the user pressing 'PRIMER' from D3PLOT or T/HIS to start PRIMER, and then passing data between the programs. If the user has not already started the remote program by doing this manually, the Link class will start the remote program.
The following simple example shows a script that is running in PRIMER, starting a link to D3PLOT. Once the link is established, the scripts running in PRIMER and D3PLOT can pass string data to the other program. This data is then displayed in a window on the remote program.
Note that just like when starting the link from PRIMER to D3PLOT manually, to start the link from the script, a model must already be loaded into PRIMER that also has results files for D3PLOT.
Script 'primer.js' running in PRIMER:

```
// Global scope for link variable
var link;
// Get path for this script, and hence path for d3plot.js
var scriptD3PLOT = arguments[0].replace('primer.js', 'd3plot.js');
// Create a simple GUI
Window.Theme(Window.THEME_CURRENT);
var w = new Window("Link PRIMER", 0.9, 1.0, 0.9, 1.0);
var start = new Widget(w, Widget.BUTTON, 1, 40, 1, 7, "Start D3PLOT");
start.onClick = startD3PLOT;
var msg  = new Widget(w, Widget.TEXTBOX,  1, 50, 8, 14, "");
var send = new Widget(w, Widget.BUTTON,  51, 90, 8, 14, "Send Message");
send.onClick = sendMessage;
send.active = false;
w.onClose = Exit;
w.Show(false);
////////////////////////////////////////////////////////////////////////////
function startD3PLOT()
{
// Function to start D3PLOT and create link
// object to pass to Link constructor
    var obj = {
               script:             scriptD3PLOT,
               connectFunction:    "connectD3PLOT",
               receiveFunction:    "receiveD3PLOT",
               disconnectFunction: "Exit",
               onReceive:          receivePRIMER,
               onDisconnect:       Exit,
              };
// Create link and save to global scope for this simple example
    link = new Link(Link.D3PLOT, obj);
// Do NOT try to immediately send something to D3PLOT using the link
// It may not be properly initialised yet.
// See functions connectD3PLOT and receivePRIMER
// Make the 'Send Message' button active and disable the 'Start D3PLOT' button
```

```
    start.active = false;
    send.active  = true;
}
////////////////////////////////////////////////////////////////////////////
function receivePRIMER(data)
{
// Function called when data is received from D3PLOT across the link
// If the object has a property called 'connect' then D3PLOT has sent
// this from the connect function and so is ready to receive data.
    if (data.connect)
    {
        Message("D3PLOT is connected");
// We could now send information to D3PLOT if needed
//      link.Send(...);
    }
// If the object has a property called 'string' then it is the string
// from the message box
    else if (data.string)
    {
        Window.Message("Example string from D3PLOT", data.string );
    }
}
////////////////////////////////////////////////////////////////////////////
function sendMessage()
{
// Sends whatever is in textbox when 'Send Message' is pressed, to D3PLOT
    var uuid = link.Send( { string: msg.text } );
}
```

Script 'd3plot.js' running in D3PLOT:

```
// Global scope for link variable
var link;
// Create a simple GUI
Window.Theme(Window.THEME_CURRENT);
var w = new Window("Link D3PLOT", 0.9, 1.0, 0.9, 1.0);
var msg  = new Widget(w, Widget.TEXTBOX,  1, 50, 1, 7, "");
var send = new Widget(w, Widget.BUTTON,  51, 90, 1, 7, "Send Message");
send.onClick = sendMessage;
send.active  = false;
w.onClose = Exit;
w.Show(false);
////////////////////////////////////////////////////////////////////////////
function connectD3PLOT(l)
{
// Function called when the connection from PRIMER is made
// Save link to global scope for this simple example
    link = l;
// Make the 'Send Message' button active
    send.active = true;
// Send a command back to PRIMER to say that we are connected
    link.Send( { connect: "CONNECTED" } );
}
////////////////////////////////////////////////////////////////////////////
function receiveD3PLOT(data)
{
// Function called when data is received from PRIMER across the link
// If the object has a property called 'string' then it is the string
// from the message box
    if (data.string)
    {
        Window.Message("Example string from PRIMER", data.string);
    }
}
////////////////////////////////////////////////////////////////////////////
function sendMessage()
{
// Sends whatever is in textbox when 'Send Message' is pressed, to PRIMER
    var uuid = link.Send( { string: msg.text } );
}
```

Constructor

new Link(program*[constant]*, data*[object]*)

Description

Create a new Link object to communicate with one of the linked Oasys LS-DYNA programs running on the computer, starting it if required.
Note that you should **not** try to use the Send method immediately after the constructor. If you do, the remote program may not have time to initialise properly. If you want to interact with the remote program immediately then the connect function in the remote program should send a command to the local program to say that it is initialised. The onReceive function in the local program can then be used to send the command. See the simple example at the top of this class for more details.
For the specific case of D3PLOT starting T/HIS, the connect function will not be called straight away. As D3PLOT and T/HIS are integrated together into a single user interface, the script **must** return back to the main event loop after the constructor is called so that D3PLOT and T/HIS can communicate with each other to set up the integration. Once this has been done and D3PLOT is 'idle', the connect function will be called. The idleTime property in the data object can be used to control this delay if required. As D3PLOT must return back to the event loop when starting T/HIS do not put any statements immediately after the link constructor call.

Arguments

- **program** (constant)

The program that you want to link to. See the programs constants

- **data** (object)

link data

Object has the following properties:

Name	Type	Description
connectFunction	string	Name of function to call in the remote script when the link connected. This function will be called when the connection is made with a Link object as the argument.
disconnectFunction (optional)	string	Name of function to call in the remote script when the link is disconnected, either because the local program has terminated, or the local script has finished. The status property can be used to distinguish these cases. This is provided as an easy way to set up a disconnect callback in the remote script. Alternatively, the remote script could set the onDisconnect property in connectFunction.
idelTime (optional)	integer	For the specific case of starting T/HIS from D3PLOT, the required time for D3PLOT to be idle for in msecs before the connect function is called. If omitted then a default idle time of 1s is used
model (optional)	integer	Model number to pass across the link to the remote program
onDisconnect (optional)	function	Function to call in the local script when the link is disconnected, either because the remote program has terminated, or the remote script has finished. The status property can be used to distinguish these cases.
onReceipt (optional)	function	Function to call in the local script to acknowledge receipt of the data in the remote program (data sent from a Send command in the local program to the remote program). The function takes a single argument which is the UUID returned from the original Send command. This function is useful if the link is asynchronous to know that the command has been processed. In the synchronous case the local program will wait for the remote program to process the Send command so this is unnecessary. However, the function can still be used.
onReceive (optional)	function	Function to call in the local script when data is received from a Send command in the remote script. The function takes a single argument which contains the data passed by Send
path (optional)	string	Path to a model to pass across the link to the remote program. This can be used instead of a model. For example if you want to open a different model in D3PLOT to what you have open in PRIMER then you could pass the path to the d3plot file to open

receiptFunction (optional)	string	Name of function to call in the remote script to acknowledge receipt of the data in the local script (data sent from a Send command in the remote link object to the local script). The function takes a single argument which is the UUID returned from the original Send command. Note that this function will only be called if the link is asynchronous. In the synchronous case the local program will wait for the remote to process the Send command so this is unnecessary. This is provided as an easy way to set up a receipt callback in the remote script. Alternatively, the remote script could set the onReceipt property in connectFunction.
receiveFunction (optional)	string	Name of function to call in the remote script when data is received from a Send command in the local script. The function takes a single argument which contains the data passed by Send. This is provided as an easy way to set up a receive callback in the remote script. Alternatively, the remote script could set the onReceive property in connectFunction.
script	string	The script to run
size (optional)	integer	Size of shared memory buffer in KB. If omitted a default size of 10KB is used
synchronous (optional)	boolean	Whether link Send commands are synchronous or asynchronous (default if omitted)
timeout (optional)	integer	Timeout for Send commands in msecs. If equal to 0 (default if omitted) then a default timeout of 5s is used

Returns

Link object

Return type

Link

Example

To create a connection to D3PLOT, calling function 'myConnectFunc' in remote script '/example/script.js' when the connection is made:

```
var s = new Link(Link.D3PLOT, { script: '/example/script.js', connectFunction: 'myConnectFunc' } );
```

Details of functions

ConnectPending() [static]

Description

Check if the script has been started on the remote side of a link and, if so, whether calling the connect function is pending. This may be useful to make a script behave differently, depending on whether it is being run normally or has been started on the remote side of a link.

Arguments

No arguments

Returns

true if a remote link script is waiting to connect, false for all other cases

Return type

boolean

Example

To check if the script has been started on the remote side of a link:

```
if (Link.ConnectPending() ) { do something ... }
```

Send(data*[ArrayBuffer or string]*)

Description

Send data/command across link

Arguments

- **data** (ArrayBuffer or string)

The data/command to pass. The data can either be an ArrayBuffer or any JavaScript type that can be represented as JSON

Returns

UUID string to uniquely identify this Send command

Return type

String

Example

To send some data across link l:

```
var data = { action: "read", filename: "example.ptf" };
l.Send(data);
```

Terminate()

Description

Terminate the program on the remote side of the link

Arguments

No arguments

Returns

No return value

Example

To terminate the program on the remote side of link l:

```
l.Terminate();
```

Material class

The Material class gives you access to materials in D3PLOT. More...

The D3PLOT JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- First(model*[Model]*)
- FlagAll(model*[Model]*, flag*[Flag]*)
- GetAll(model*[Model]*, property (optional)*[string]*)
- GetFlagged(model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(model*[Model]*, label*[integer]*)
- GetFromIndex(model*[Model]*, index*[integer]*)
- Last(model*[Model]*)
- Total(model*[Model]*)
- UnflagAll(model*[Model]*, flag*[Flag]*)

Member functions

- ClearFlag(flag*[Flag]*)
- Flagged(flag*[Flag]*)
- Next()
- Previous()
- SetFlag(flag*[Flag]*)

Material properties

Name	Type	Description
include (read only)	integer	The include file number in the model that the material is in
index (read only)	integer	The internal index for the material in D3PLOT (starting at 0)
label (read only)	integer	The Ansys LS-DYNA label for the material
model (read only)	Model	The Model that the material is in
name (read only)	string	The name for the material type (e.g. *MAT_RIGID)
title (read only)	string	The title for the material (or null if no title)
type (read only)	constant	The type for the material (will be Type.MATERIAL)

Detailed Description

The Material class allows you to inspect materials in a model. This information will only be available if a ztf file has been read for a model. See the documentation below for more details.

Details of functions

ClearFlag(flag*[Flag]*)

Description

Clears a flag on a material

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to clear on the material

Returns

No return value

Example

To clear flag f on material m:

```
m.ClearFlag();
```

First(model*[Model]*) [static]

Description

Returns the first material in the model (or null if there are no materials in the model)

Arguments

- **model** (Model)

Model to get first material in

Returns

Material object

Return type

Material

Example

To get the first material in model m:

```
var m = Material.First(m);
```

FlagAll(model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the materials in the model with a defined flag

Arguments

- **model** (Model)

Model that all the materials will be flagged in

- **flag** (Flag)

Flag (see AllocateFlag) to set on the materials

Returns

No return value

Example

To flag all of the materials with flag f in model m:

```
Material.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the material is flagged or not

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to test on the material

Returns

true if flagged, false if not

Return type

boolean

Example

To check if material m has flag f set on it:

```
if (m.Flagged(f) ) do_something...
```

GetAll(model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Material objects or properties for all of the materials in the model. If the optional property argument is not given then an array of Material objects is returned. If the property argument is given, that property value for each material is returned in the array instead of a Material object

Arguments

- **model** (Model)

Model that all the materials are in

- **property (optional)** (string)

Name for property to get for all materials in the model

Returns

Array of Material objects or properties

Return type

Array

Example

To get all of the materials in model m:

```
var m = Material.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'label' for a material) for each material in model m:

```
var a = Material.GetAll(m, 'foo');
```

GetFlagged(model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Gets all of the materials in the model flagged with a defined flag. If the optional property argument is not given then an array of Material objects is returned. If the property argument is given, that property value for each material is returned in the array instead of a Material object

Arguments

- **model** (Model)

Model that the flagged materials are in

- **flag** (Flag)

Flag (see AllocateFlag) set on the materials to get

- **property (optional)** (string)

Name for property to get for all flagged materials in the model

Returns

Array of Material objects or properties

Return type

Array

Example

To get all of the materials flagged with flag f in model m:

```
Material.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'label' for a node) for all materials flagged with flag f in model m:

```
var a = Material.GetFlagged(m, f, 'foo');
```

GetFromID(model*[Model]*, label*[integer]*) [static]

Description

Returns the Material object for material in model with label (or null if it does not exist)

Arguments

- **model** (Model)

Model to get material in

- **label** (integer)

The Ansys LS-DYNA label for the material in the model

Returns

Material object

Return type

Material

Example

To get the material in model m with label 1000:

```
var m = Material.GetFromID(m, 1000);
```

GetFromIndex(model*[Model]*, index*[integer]*) [static]

Description

Returns the Material object for material in model with index (or null if it does not exist)

Arguments

- **model** (Model)

Model to get material in

- **index** (integer)

The D3PLOT internal index in the model for material, starting at 0.

Returns

Material object

Return type

Material

Example

To get the 51st material in model m:

```
var m = Material.GetFromIndex(m, 50);
```

Last(model*[Model]*) [static]

Description

Returns the last material in the model (or null if there are no materials in the model)

Arguments

- **model** (Model)

Model to get last material in

Returns

Material object

Return type

Material

Example

To get the last material in model m:

```
var m = Material.Last(m);
```

Next()

Description

Returns the next material in the model (or null if there is not one)

Arguments

No arguments

Returns

Material object

Return type

Material

Example

To get the next material after material m:

```
m = m.Next();
```

Previous()

Description

Returns the previous material in the model (or null if there is not one)

Arguments

No arguments

Returns

Material object

Return type

Material

Example

To get the previous material before material m:

```
m = m.Previous();
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on a material

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to set on the material

Returns

No return value

Example

To set flag f on material m:

```
m.SetFlag(f);
```

Total(model*[Model]*) [static]

Description

Returns the total number of materials in the model

Arguments

- **model** (Model)

Model to get total in

Returns

The number of materials

Return type

integer

Example

To get the number of materials in model m:

```
var total = Material.Total(m);
```

UnflagAll(model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the materials in the model

Arguments

- **model** (Model)

Model that the defined flag for all materials will be unset in

- **flag** (Flag)

Flag (see AllocateFlag) to unset on the materials

Returns

No return value

Example

To unset flag f on all of the materials in model m:

```
Material.UnflagAll(m, f);
```

Measure class

The Measure class allows you to create measurements between nodes and parts. More...

The D3PLOT JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Delete(type*[integer]*)
- DeleteAll()
- GetFromIndex(index*[integer]*)

Member functions

- Data()

Measure constants

Constants for Measure type

Name	Description
Measure.NODE_ANGLE	Node angle measure type
Measure.NODE_ORIGIN	Node to origin measure type
Measure.NODE_TO_NODE	Node to node measure type
Measure.NODE_TO_PART	Node to part measure type
Measure.PART_TO_PART	Part to part measure type

Constants for display

Name	Description
Measure.LABEL	Display measure entity labels
Measure.MAGNITUDE	Display measurement magnitude
Measure.TRANSPARENT	Transparent display of measurements
Measure.VECTOR	Display measurement vector

Constants for line

Name	Description
Measure.HIDDEN	Hidden line style for measures
Measure.WIREFRAME	Wireframe line style for measures

Constants for number format

Name	Description
Measure.AUTOMATIC	Automatic number format
Measure.GENERAL	General number format, for example 123.4
Measure.SCIENTIFIC	Scientific number format, for example, 1.234E+2

Constants for offset

Name	Description
Measure.DEFORM	Consider Deform (Shift deform, Magnify etc.) coordinates for measure calculations
Measure.MODEL_SPACE	Consider model space offsets in the graphics window

Constants for show

Name	Description
Measure.SHOW_ALL	Show all measures
Measure.SHOW_CURRENT	Show only the current measure
Measure.SHOW_NONE	Show no measure

Measure class properties

Name	Type	Description
consider_ blanking	logical	true if calculations in Measure.Data() only consider unblanked elements, false (default) if they consider all elements.
current	integer	Index of the measure currently selected for display purposes, starting at 1.
display	constant	Display options for measurements. Can be any combination (bitwise OR) of Measure.MAGNITUDE, Measure.VECTOR, Measure.LABEL and Measure.TRANSPARENT.
format	constant	Number format for measures. Can be Measure.AUTOMATIC, Measure.SCIENTIFIC or Measure.GENERAL.
line	constant	Line style for measures. Can be Measure.HIDDEN or Measure.WIREFRAME.
offsets	constant	Whether or not to consider Deform (Shift deform, Magnify etc.) or model space offsets in the calculation of measures. Can be any combination (bitwise OR) of Measure.DEFORM, Measure.MODEL_SPACE.
precision	integer	Number of decimal places for scientific or general number format.
show	constant	Whether all, only the current or no measure is shown. Can be Measure.SHOW_NONE, Measure.SHOW_CURRENT or Measure.SHOW_ALL.
total (read only)	integer	Total number of measures in D3PLOT

Measure properties

Name	Type	Description
index (read only)	integer	Index where the measure appears on the interactive measure menu, starting at 1. Note that the index of a measure can change when measures with lower indices are deleted.
name	string	Measure name
type (read only)	constant	Measure type. Can be Measure.NODE_TO_NODE, Measure.NODE_ANGLE, Measure.NODE_ORIGIN, Measure.NODE_TO_PART or Measure.PART_TO_PART.

Detailed Description

The Measure class allows you to measure distances and angles between nodes and/or parts. The following example shows how Measures are used to get the distance between two parts.

```
// Create a measure between parts 21 and 22 in model 1:
var mod = Model.GetFromID(1);
var p1 = Part.GetFromID(mod, 21);
var p2 = Part.GetFromID(mod, 22);
var m = new Measure(Measure.PART_TO_PART, { part1: p1, part2: p2});
// Get the distace between these parts at state 10:
mod.state = 10;
var data = m.Data();
var distance = data.mag;
// Show all measurements on the screen in state 20:
// Note that the Model 'state' property would only affect model data
// handling, but here we are interested in the state on the screen, so
// we set it with 'state' on the GraphicsWindow object instead. The
// following assumes that we only have got one graphics window W1, but
// loops over multiple graphics windows would be possible too.
var gw = GraphicsWindow.GetFromID(1);
gw.state = 20;
Measure.show = Measure.SHOW_ALL;
```

See the documentation below for more details.

Constructor

new Measure(type*[constant]*, options*[object]*)

Description

Create a new Measure object.

Arguments

- **type** (constant)

Measure type. Can be Measure.NODE_TO_NODE, Measure.NODE_ANGLE, Measure.NODE_ORIGIN, Measure.NODE_TO_PART or Measure.PART_TO_PART

- **options** (object)

Measure options.

Object has the following properties:

Name	Type	Description
node1	Node object	First node for the measurement
node2	Node object	Second node for the measurement
node3	Node object	Third node for the measurement
part1	Part object	First part for the measurement
part2	Part object	Second part for the measurement
window	GraphicsWindow object	Graphics window where the measurement will be created. If omitted, it will be created in the first window.

Returns

Measure object

Return type

Measure

Example

To create a measure between the node with external label 50 in model 1 and the part with external label 10 in model 2 in window 4:

```
var model1 = Model.GetFromID(1);
var node = Node.GetFromID(model1, 50);
var model2 = Model.GetFromID(2);
var part = Part.GetFromID(model2, 10);
var gw = GraphicsWindow.GetFromID(4);
var m = new Measure(Measure.NODE_TO_PART, { node1: node, part2: part, window: gw
});
```

new Measure(type*[constant]*, options*[object]*) [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Create a new Measure object.

Arguments

- **type** (constant)

Measure type. Can be Measure.NODE_TO_NODE, Measure.NODE_ANGLE, Measure.NODE_ORIGIN, Measure.NODE_TO_PART or Measure.PART_TO_PART

- **options** (object)

Measure options.

Object has the following properties:

Name	Type	Description
model1	integer	Model ID for the first node or part for the measurement. If omitted, the current model will be used.
model2	integer	Model ID for the second node or part for the measurement. If omitted, the current model will be used.
model3	integer	Model ID for the third node for the measurement. If omitted, the current model will be used.
node1	integer	First node for the measurement. If +ve: internal node number starting at 1 (faster), if -ve: external node label.
node2	integer	Second node for the measurement. If +ve: internal node number starting at 1 (faster), if -ve: external node label.
node3	integer	Third node for the measurement. If +ve: internal node number starting at 1 (faster), if -ve: external node label.
part1	integer	First part for the measurement If +ve: internal part number starting at 1 (faster), if -ve: external part label.
part2	integer	Second part for the measurement. If +ve: internal part number starting at 1 (faster), if -ve: external part label.
window	integer	Graphics window ID where the measurement will be created. If omitted, it will be created in the first window.

Returns

Measure object

Return type

Measure

Example

To create a measure between the 100th node and the 3rd part in the model:

```
var m = new Measure(Measure.NODE_TO_PART, { node1: 100, part2: 3 });
```

To create a measure between the node with external label 50 and the part with external label 10:

```
var m = new Measure(Measure.NODE_TO_PART, { node1: -50, part2: -10 });
```

Details of functions

Data()

Description

Returns an object with data for this measure.

Arguments

No arguments

Returns

Object with the following properties:

Name	Type	Description
dx	real	X component of the measurement (for types Measure.NODE_TO_NODE, Measure.NODE_ORIGIN, Measure.NODE_TO_PART or Measure.PART_TO_PART)
dy	real	Y component of the measurement (for types Measure.NODE_TO_NODE, Measure.NODE_ORIGIN, Measure.NODE_TO_PART or Measure.PART_TO_PART)
dz	real	Z component of the measurement (for types Measure.NODE_TO_NODE, Measure.NODE_ORIGIN, Measure.NODE_TO_PART or Measure.PART_TO_PART)
mag	real	magnitude of the measurement (length for types Measure.NODE_TO_NODE, Measure.NODE_ORIGIN, Measure.NODE_TO_PART or Measure.PART_TO_PART, angle in degrees for type Measure.NODE_ANGLE)
xy	real	Angle in the XY plane in degrees for type Measure.NODE_ANGLE
xz	real	Angle in the XZ plane in degrees for type Measure.NODE_ANGLE. The zx property is an alternative name for this.
yz	real	Angle in the YZ plane in degrees for type Measure.NODE_ANGLE
zx	real	Angle in the ZX plane in degrees for type Measure.NODE_ANGLE. The xz property is an alternative name for this.

Return type

object

Example

To get the X component of the measure object m:

```
var data = m.Data();
var dx = data.dx;
```

Delete(type*[integer]*) [static]

Description

Deletes the measure with the given index.

Arguments

- **type** (integer)

Index of the measure to be deleted, starting at 1.

Returns

No return value

Example

To delete measure m:

```
var index = m.index;
Measure.Delete(index);
```

DeleteAll() [static]

Description

Deletes all measures.

Arguments

No arguments

Returns

No return value

Example

To delete all measures:

```
Measure.DeleteAll();
```

GetFromIndex(index*[integer]*) [static]

Description

Gets the measure object for a given index.

Arguments

- **index** (integer)

Index of the measure, starting at 1.

Returns

Measure object for that index

Return type

Measure

Example

To get the object for the measure at index 3:

```
var m = Measure.GetFromIndex(3);
```

Model class

The Model class gives you access to models in D3PLOT. More...

The D3PLOT JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- First()
- GetFromID(model number*[integer]*)
- Highest()
- Last()
- Read(filename*[string]*)
- Total()

Member functions

- ClearFlag(flag*[Flag]*)
- Delete()
- GraphicsWindows()
- Next()
- Previous()
- ReadPropertiesFile(filename*[string]*, info (optional)*[object]*)
- Reread()
- Rescan()
- Time(state*[integer]*)

Model properties

Name	Type	Description
filename (read only)	boolean	The model filename
number (read only)	integer	The model number
state	integer	The state in the model used for scripting methods. Note that this is **not** the state that is displayed for a model in a graphics window. This property is only used for some of the scripting methods. Many of the methods in the API depend on which state the model is in and setting this property alters that state. To set the state used for all the "get" and "put" functions which handle model-related data use this property. To change the state that is displayed for a model in a graphics window use the GraphicsWindow state property
states	integer	The total number of states in the model
title	string	The model title

Detailed Description

The Model class allows you to do various operations on models in D3PLOT. There are various methods and properties available that allow you do read and operate models. See the documentation below for more details.

Constructor

new Model(filename*[string]*)

Description

Reads a file into the first free model in D3PLOT

Arguments

- **filename** (string)

Filename you want to read

Returns

Model object

Return type

Model

Example

To create a model in D3PLOT for the file /data/test/file.ptf

```
var m = new Model("/data/test/file.ptf");
```

Details of functions

ClearFlag(flag*[Flag]*)

Description

Clears a flag on all of the items in the model

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to clear

Returns

No return value

Example

To clear flag f on all items in model m:

```
m.ClearFlag(f);
```

Delete()

Description

Deletes a model in D3PLOT
Do not use the Model object after calling this method.

Arguments

No arguments

Returns

No return value

Example

To delete model m in D3PLOT

```
m.Delete();
```

First() [static]

Description

Returns the Model object for the first model in D3PLOT (or null if there are no models)

Arguments

No arguments

Returns

Model object

Return type

Model

Example

To get the Model object for the first model:

```
var m = Model.First();
```

GetFromID(model number*[integer]*) [static]

Description

Returns the Model object for a model ID (or null if model does not exist)

Arguments

- **model number** (integer)

number of the model you want the Model object for

Returns

Model object

Return type

Model

Example

To get the Model object for model number 1

```
var m = Model.GetFromID(1);
```

GraphicsWindows()

Description

Returns the graphics window(s) that the model exists in

Arguments

No arguments

Returns

Array of GraphicsWindow objects

Return type

array

Example

To get the graphics windows model m exists in:

```
var list = m.GraphicsWindows();
```

Highest() [static]

Description

Returns the highest model number in D3PLOT (or 0 if no models). Also see Total()

Arguments

No arguments

Returns

Highest model number

Return type

integer

Example

To get the highest model number:

```
var highest = Model.Highest();
```

Last() [static]

Description

Returns the Model object for the last model in D3PLOT (or null if there are no models)

Arguments

No arguments

Returns

Model object

Return type

Model

Example

To get the Model object for the last model:

```
var m = Model.Last();
```

Next()

Description

Returns the next model (or null if there is not one)

Arguments

No arguments

Returns

Model object

Return type

Model

Example

To get the model after model m:

```
m = m.Next();
```

Previous()

Description

Returns the previous model (or null if there is not one)

Arguments

No arguments

Returns

Model object

Return type

Model

Example

To get the model before model m:

```
m = m.Previous();
```

Read(filename*[string]*) [static]

Description

Reads a file into D3PLOT

Arguments

- **filename** (string)

Filename you want to read

Returns

Model object

Return type

Model

Example

To create a model in D3PLOT from the file /data/test/file.ptf

```
var m = Model.Read("/data/test/file.ptf");
```

ReadPropertiesFile(filename*[string]*, info (optional)*[object]*)

Description

Reads a properties file for the model

Arguments

- **filename** (string)

Filename for the properties file you want to read

- **info (optional)** (object)

Object containing the information to set. Can be any of:

Object has the following properties:

Name	Type	Description
ignoreElements	boolean	Ignore any element properties in the properties file and only process part based entries (default is false)
preBlank	boolean	Blank everything in the model before reading the properties file (default is false)

Returns

No return value

Example

To read the properties file /data/test/my_properties.prp for model m:

```
m.ReadPropertiesFile("/data/test/my_properties.prp");
```

Reread()

Description

Rereads the model

Arguments

No arguments

Returns

No return value

Example

To reread model m:

```
m.Reread();
```

Rescan()

Description

Rescans the model

Arguments

No arguments

Returns

No return value

Example

To rescan model m:

```
m.Rescan();
```

Time(state*[integer]*)

Description

Returns the analysis time for a particular state in the model

Arguments

- **state** (integer)

The state you want to get the time for (0 <= state <= states)

Returns

Analysis time

Return type

real

Example

To get the analysis time for state 10 in model m:

```
var time = m.Time(10);
```

Total() [static]

Description

Returns the total number of models in use in D3PLOT. Also see Highest()

Arguments

No arguments

Returns

Total number of models in use

Return type

integer

Example

To get total number of models:

```
var total = Model.Total();
```

Node class

The Node class gives you access to nodes in D3PLOT. More...

The D3PLOT JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(window*[GraphicsWindow]*, model*[Model]*)
- BlankFlagged(window*[GraphicsWindow]*, model*[Model]*, flag*[Flag]*)
- First(model*[Model]*)
- FlagAll(model*[Model]*, flag*[Flag]*)
- GetAll(model*[Model]*, property (optional)*[string]*)
- GetFlagged(model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(model*[Model]*, label*[integer]*)
- GetFromIndex(model*[Model]*, index*[integer]*)
- GetMultipleData(component*[constant]*, items*[array]*, options (optional)*[object]*)
- Last(model*[Model]*)
- Pick()
- Select(flag*[Flag]*)
- Total(model*[Model]*)
- UnblankAll(window*[GraphicsWindow]*, model*[Model]*)
- UnblankFlagged(window*[GraphicsWindow]*, model*[Model]*, flag*[Flag]*)
- UnflagAll(model*[Model]*, flag*[Flag]*)

Member functions

- Acceleration()
- Blank(window*[GraphicsWindow]*)
- Blanked(window*[GraphicsWindow]*)
- ClearFlag(flag*[Flag]*)
- Coordinates()
- Displacement()
- Elements(type*[constant]*)
- Flagged(flag*[Flag]*)
- GetData(component*[constant]*, options (optional)*[object]*)
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Unblank(window*[GraphicsWindow]*)
- Velocity()

Node properties

Name	Type	Description
colour	Colour	The colour of the node
data (read only)	real|array	Component data for a node passed as an argument to GetMultipleData. Note that data will only exist for the instance of the node passed to GetMultipleData. i.e. it is a local property stored on the specific instance. It is not stored in the D3PLOT database
displayMode	integer	The display mode of the node. Can be Constant.WIREFRAME, Constant.HIDDEN, Constant.SHADED or Constant.CURRENT.
include (read only)	integer	The include file number in the model that the node is in

index (read only)	integer	The internal index for the node in D3PLOT (starting at 0)
label (read only)	integer	The Ansys LS-DYNA label for the node
model (read only)	Model	The Model that the node is in
overlayColour	Colour	The overlay colour of the node
overlayMode	integer	The overlay mode of the node. Can be Constant.NO_OVERLAY, Constant.FREE_EDGE, Constant.ALL_EDGES or Constant.CURRENT.
transparency	integer	The transparency of the node between 0 (opaque) and 100 (clear)
type (read only)	constant	The type for the node (will be Type.NODE)

Detailed Description

The Node class allows you to inspect nodes in a model. See the documentation below for more details.

Details of functions

Acceleration()

Description

Returns the acceleration for the node

Arguments

No arguments

Returns

Array containing the nodal acceleration [Ax, Ay, Az] (or null if the value cannot be calculated)

Return type

array

Example

To return the acceleration of node n:

```
var acc = n.Acceleration();
if (acc !== null) do_something...
```

Blank(window*[GraphicsWindow]*)

Description

Blanks the node in a graphics window

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to blank the node in

Returns

No return value

Example

To blank node n in graphics window g:

```
n.Blank(g);
```

BlankAll(window*[GraphicsWindow]*, model*[Model]*) [static]

Description

Blanks all of the nodes in the model

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to blank the nodes in

- **model** (Model)

Model that all the nodes will be blanked in

Returns

No return value

Example

To blank all of the nodes in model m, in graphics window gw:

```
Node.BlankAll(gw, m);
```

BlankFlagged(window*[GraphicsWindow]*, model*[Model]*, flag*[Flag]*) [static]

Description

Blanks all of the nodes in the model flagged with a defined flag

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to blank the nodes in

- **model** (Model)

Model that the flagged nodes will be blanked in

- **flag** (Flag)

Flag (see AllocateFlag) set on the nodes to blank

Returns

No return value

Example

To blank all of the nodes flagged with flag f in model m, in graphics window gw:

```
Node.BlankFlagged(gw, m, f);
```

Blanked(window*[GraphicsWindow]*)

Description

Checks if the node is blanked in a graphics window or not

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) in which to check if the node is blanked

Returns

true if blanked, false if not

Return type

boolean

Example

To check if node n is blanked in graphics window g:

```
if (n.Blanked(g) ) do_something...
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on a node

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to clear on the node

Returns

No return value

Example

To clear flag f on node n:

```
n.ClearFlag();
```

Coordinates()

Description

Returns the coordinates for the node

Arguments

No arguments

Returns

Array containing the nodal coordinates [Cx, Cy, Cz] (or null if the value cannot be calculated)

Return type

array

Example

To return the coordinates of node n:

```
var coords = n.Coordinates();
if (coords !== null) do_something...
```

Displacement()

Description

Returns the displacement for the node

Arguments

No arguments

Returns

Array containing the nodal displacement [Dx, Dy, Dz] (or null if the value cannot be calculated)

Return type

array

Example

To return the displacement of node n:

```
var disp = n.Displacement();
if (disp !== null) do_something...
```

Elements(type*[constant]*)

Description

Returns the elements using this node

Arguments

- **type** (constant)

The type of elements. Either Type.BEAM, Type.SHELL, Type.SOLID or Type.TSHELL

Returns

Array containing the elements or null if there are no elements

Return type

array

Example

To return the shell elements using node n:

```
var shells = n.Elements(Type.SHELL);
```

First(model*[Model]*) [static]

Description

Returns the first node in the model (or null if there are no nodes in the model)

Arguments

- **model** (Model)

Model to get first node in

Returns

Node object

Return type

Node

Example

To get the first node in model m:

```
var n = Node.First(m);
```

FlagAll(model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the nodes in the model with a defined flag

Arguments

- **model** (Model)

Model that all the nodes will be flagged in

- **flag** (Flag)

Flag (see AllocateFlag) to set on the nodes

Returns

No return value

Example

To flag all of the nodes with flag f in model m:

```
Node.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the node is flagged or not

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to test on the node

Returns

true if flagged, false if not

Return type

boolean

Example

To check if node n has flag f set on it:

```
if (n.Flagged(f) ) do_something...
```

GetAll(model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Node objects or properties for all of the nodes in the model. If the optional property argument is not given then an array of Node objects is returned. If the property argument is given, that property value for each node is returned in the array instead of a Node object

Arguments

- **model** (Model)

Model that all the nodes are in

- **property (optional)** (string)

Name for property to get for all nodes in the model

Returns

Array of Node objects or properties

Return type

Array

Example

To get all of the nodes in model m:

```
var n = Node.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'label' for a node) for each node in model m:

```
var a = Node.GetAll(m, 'foo');
```

GetData(component*[constant]*, options (optional)*[object]*)

Description

Returns the value for a data component.
Also see GetMultipleData

Arguments

- **component** (constant)

Component constant to get data for

- **options (optional)** (object)

Object containing options for getting data. Can be any of:

Object has the following properties:

Name	Type	Description
extra	integer	The extra data component number if component Component.SOX for solids, Component.BMX for beams or Component.SHX for shells and thick shells
ip	integer	Integration point number to get the data at (ip >= 1 or one of the constants Constant.TOP, Constant.MIDDLE or Constant.BOTTOM). If the integration point is not defined it will use the integration point defined on the current GUI "data" panel, which defaults to the middle surface for shells, thick shells, and solids, and Mag All for beams, but may vary if changed by an interactive user. If consistent output from a script is required, independent of any prior interactive activity, an explicit integration point or surface should be defined
op	integer	On plane integration point number for shells and thick shells (op >= 1 [default])

referenceFrame	constant	The frame of reference to return values in. Either Constant.GLOBAL (default), Constant.LOCAL, Constant.CYLINDRICAL, Constant.USER_DEFINED or Constant.MATERIAL. This is only necessary for directional components (eg X stress) and then only when something other than the default Constant.GLOBAL coordinate system is to be used
user	integer	The user-defined component number if component Component.UNOS, Component.UNOV, Component.USSS, Component.USST, Component.UBMS or Component.UBMV

Returns

Number if a scalar component, array if a vector or tensor component (or null if the value cannot be calculated because it's not available in the model).
If requesting an invalid component it will throw an error (e.g. Component.AREA of a node).

Return type

real|array

Example

To calculate a component and check it has been calculated (note that in the example, the argument extra is optional):

```
var value = n.GetData(component, {extra: 1});
if (value !== null) do_something...
```

GetFlagged(model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Gets all of the nodes in the model flagged with a defined flag. If the optional property argument is not given then an array of Node objects is returned. If the property argument is given, that property value for each node is returned in the array instead of a Node object

Arguments

- **model** (Model)

Model that the flagged nodes are in

- **flag** (Flag)

Flag (see AllocateFlag) set on the nodes to get

- **property (optional)** (string)

Name for property to get for all flagged nodes in the model

Returns

Array of Node objects or properties

Return type

Array

Example

To get all of the nodes flagged with flag f in model m:

```
Node.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'label' for a node) for all nodes flagged with flag f in model m:

```
var a = Node.GetFlagged(m, f, 'foo');
```

GetFromID(model*[Model]*, label*[integer]*) [static]

Description

Returns the Node object for node in model with label (or null if it does not exist)

Arguments

- **model** (Model)

Model to get node in

- **label** (integer)

The Ansys LS-DYNA label for the node in the model

Returns

Node object

Return type

Node

Example

To get the node in model m with label 1000:

```
var n = Node.GetFromID(m, 1000);
```

GetFromIndex(model*[Model]*, index*[integer]*) [static]

Description

Returns the Node object for node in model with index (or null if it does not exist)

Arguments

- **model** (Model)

Model to get node in

- **index** (integer)

The D3PLOT internal index in the model for node, starting at 0.

Returns

Node object

Return type

Node

Example

To get the 51st node in model m:

```
var n = Node.GetFromIndex(m, 50);
```

GetMultipleData(component*[constant]*, items*[array]*, options (optional)*[object]*) [static]

Description

Returns the value for a data component for multiple nodes. For each node a local property called data will be created containing a number if a scalar component, or an array if a vector or tensor component (or null if the value cannot be calculated). The data is also returned as an object.
Also see GetData

Arguments

- **component** (constant)

Component constant to get data for

- **items** (array)

Array of Node objects to get the data for. All of the nodes must be in the same model.

- **options (optional)** (object)

Object containing options for getting data. Can be any of:

Object has the following properties:

Name	Type	Description
extra	integer	The extra data component number if component Component.SOX for solids, Component.BMX for beams or Component.SHX for shells and thick shells
ip	integer	Integration point number to get the data at (ip >= 1 or one of the constants Constant.TOP, Constant.MIDDLE or Constant.BOTTOM)
op	integer	On plane integration point number for shells and thick shells (op >= 1 [default])
referenceFrame	constant	The frame of reference to return values in. Either Constant.GLOBAL (default), Constant.LOCAL, Constant.CYLINDRICAL, Constant.USER_DEFINED or Constant.MATERIAL. This is only necessary for directional components (eg X stress) and then only when something other than the default Constant.GLOBAL coordinate system is to be used
user	integer	The user-defined component number if component Component.UNOS, Component.UNOV, Component.USSS, Component.USST, Component.UBMS or Component.UBMV

Returns

Object containing the data. A property is created in the object for each node with the label. The value of the property is a number if a scalar component or an array if a vector or tensor component (or null if the value cannot be calculated)

Return type

object

Example

To calculate a component for nodes in array items and use the data property (note that in the example, the argument extra is optional):

```
Node.GetMultipleData(component, items, {extra: 1});
for (i=0; i<items.length; i++)
{
    if (items[i].data !== null) do_something...
}
```

To calculate a component for nodes in array items and use the return value (note that in the example, the argument extra is optional):

```
var data = Node.GetMultipleData(component, items, {extra: 1});
for (d in data)
{
    Message("Label is " + d);
    if (data[d] !== null) do_something...
}
```

Last(model[*Model*]) [static]

Description

Returns the last node in the model (or null if there are no nodes in the model)

Arguments

- **model** (Model)

Model to get last node in

Returns

Node object

Return type

Node

Example

To get the last node in model m:

```
var n = Node.Last(m);
```

Next()

Description

Returns the next node in the model (or null if there is not one)

Arguments

No arguments

Returns

Node object

Return type

Node

Example

To get the next node after node n:

```
n = n.Next();
```

Pick() [static]

Description

Allows the user to pick a node from the screen

Arguments

No arguments

Returns

Node object or null if cancelled

Return type

Node

Example

To pick a node:

```
var n = Node.Pick();
```

Previous()

Description

Returns the previous node in the model (or null if there is not one)

Arguments

No arguments

Returns

Node object

Return type

Node

Example

To get the previous node before node n:

```
n = n.Previous();
```

Select(flag*[Flag]*) [static]

Description

Selects nodes using an object menu

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to use when selecting nodes

Returns

The number of nodes selected or null if menu cancelled

Return type

integer

Example

To select nodes, flagging those selected with flag f:

```
var total = Node.Select(f);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on a node

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to set on the node

Returns

No return value

Example

To set flag f on node n:

```
n.SetFlag(f);
```

Total(model*[Model]*) [static]

Description

Returns the total number of nodes in the model

Arguments

- **model** (Model)

Model to get total in

Returns

The number of nodes

Return type

integer

Example

To get the number of nodes in model m:

```
var total = Node.Total(m);
```

Unblank(window*[GraphicsWindow]*)

Description

Unblanks the node in a graphics window

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to unblank the node in

Returns

No return value

Example

To unblank node n in graphics window g:

```
n.Unblank(g);
```

UnblankAll(window*[GraphicsWindow]*, model*[Model]*) [static]

Description

Unblanks all of the nodes in the model

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to unblank the nodes in

- **model** (Model)

Model that all the nodes will be unblanked in

Returns

No return value

Example

To unblank all of the nodes in model m, in graphics window gw:

```
Node.UnblankAll(gw, m);
```

UnblankFlagged(window*[GraphicsWindow]*, model*[Model]*, flag*[Flag]*) [static]

Description

Unblanks all of the nodes in the model flagged with a defined flag

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to unblank the nodes in

- **model** (Model)

Model that the flagged nodes will be unblanked in

- **flag** (Flag)

Flag (see AllocateFlag) set on the nodes to unblank

Returns

No return value

Example

To unblank all of the nodes flagged with flag f in model m, in graphics window gw:

```
Node.UnblankFlagged(gw, m, f);
```

UnflagAll(model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the nodes in the model

Arguments

- **model** (Model)

Model that the defined flag for all nodes will be unset in

- **flag** (Flag)

Flag (see AllocateFlag) to unset on the nodes

Returns

No return value

Example

To unset flag f on all of the nodes in model m:

```
Node.UnflagAll(m, f);
```

Velocity()

Description

Returns the velocity for the node

Arguments

No arguments

Returns

Array containing the nodal velocity [Vx, Vy, Vz] (or null if the value cannot be calculated)

Return type

array

Example

To return the velocity of node n:

```
var vel = n.Velocity();
if (disp !== null) do_something...
```

Options class

The Options class enables you to access several options in D3PLOT. More...

The D3PLOT JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Options class properties

Name	Type	Description
auto_ confirm	logical	If true then D3PLOT will automatically confirm (i.e. press the OK button) on (most) message boxes that are mapped. If false (default) then the message boxes will be shown and wait for the user to press a button. This option may be useful to help automate an operation where D3PLOT would normally show a message box and wait for the user to press a button.
exception_ messages	logical	If true (default) error messages will be printed to the dialogue box/stdout when an exception occurs in the API. If false they will not be printed. This option may be useful if you are using try/catch to manage exceptions and you do not want any error messages to be printed.

Properties for ssh

Name	Type	Description
ssh_ buffer_ size	integer	The size of the buffer used (in kiloBytes) when transferring data to/from the remote machine in the Ssh class. Depending on your network and the size of the files you are transferring, changing this value may make file transfers quicker. The default value is 64(kB) but any value in the range 1(kB) to 1024(kB) is allowed.

Properties for widgets

Name	Type	Description
max_widgets	integer	The maximum number of Widgets that can be made for one Window. The default value is 1000
max_ window_ lines	integer	The maximum number of lines that can be made for a Window.Error(), Window.Information(), Window.Message(), Window.Question() or Window.Warning() window. The default value is 25

Detailed Description

The Options class is used to get/set options that D3PLOT uses for certain functions. The options are available as **class** properties. See the documentation for more details. An example: Options.mass_properties_include_attached_mass_ deformable_elems=true

Page class

The Page class gives you access to pages in D3PLOT. More...

The D3PLOT JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Current()
- First()
- GetFromID(page number*[integer]*)
- Last()
- Total()

Member functions

- Next()
- Previous()
- Show()

Page constants

Constants for Layout

Name	Description
Page.LAYOUT_1_1	Layout with 1x1 windows per page
Page.LAYOUT_2_2	Layout with 2x2 windows per page
Page.LAYOUT_3_3	Layout with 3x3 windows per page
Page.LAYOUT_CUSTOM	Custom layout with user defined numer of windows per page
Page.LAYOUT_TILE_TALL	Layout with tall tiles
Page.LAYOUT_TILE_WIDE	Layout with wide tiles

Page properties

Name	Type	Description
layout	constant	The page layout. See Page layout constants
number (read only)	integer	The page number
x	integer	The number of windows in X for the LAYOUT_CUSTOM case
y	integer	The number of windows in Y for the LAYOUT_CUSTOM case

Detailed Description

The Page class allows you to modify pages in D3PLOT. There are various methods and properties available that allow you do alter how windows are displayed on pages. See the documentation below for more details.

Details of functions

Current() [static]

Description

Returns the Page object for the page currently shown in D3PLOT

Arguments

No arguments

Returns

Page object

Return type

Page

Example

To get the Page object for the current page:

```
var p = Page.Current();
```

First() [static]

Description

Returns the Page object for the first page in D3PLOT

Arguments

No arguments

Returns

Page object

Return type

Page

Example

To get the Page object for the first page:

```
var p = Page.First();
```

GetFromID(page number*[integer]*) [static]

Description

Returns the Page object for a page ID

Arguments

- **page number** (integer)

number of the page you want the Page object for

Returns

Page object

Return type

Page

Example

To get the Page object for page number 1

```
var p = Page.GetFromID(1);
```

Last() [static]

Description

Returns the Page object for the last page in D3PLOT

Arguments

No arguments

Returns

Page object

Return type

Page

Example

To get the Page object for the last page:

```
var p = Page.Last();
```

Next()

Description

Returns the next page (or null if there is not one)

Arguments

No arguments

Returns

Page object

Return type

Page

Example

To get the page after page p:

```
p = p.Next();
```

Previous()

Description

Returns the previous page (or null if there is not one)

Arguments

No arguments

Returns

Page object

Return type

Page

Example

To get the page before page p:

```
p = p.Previous();
```

Show()

Description

Shows this page in D3PLOT

Arguments

No arguments

Returns

No return value

Example

To show page p:

```
p.Show();
```

Total() [static]

Description

Returns the total number of pages in D3PLOT.

Arguments

No arguments

Returns

Total number of pages

Return type

integer

Example

To get total number of pages:

```
var total = Page.Total();
```

Part class

The Part class gives you access to parts in D3PLOT. More...

The D3PLOT JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(window*[GraphicsWindow]*, model*[Model]*)
- BlankFlagged(window*[GraphicsWindow]*, model*[Model]*, flag*[Flag]*)
- First(model*[Model]*)
- FlagAll(model*[Model]*, flag*[Flag]*)
- GetAll(model*[Model]*, property (optional)*[string]*)
- GetConditionParts(options*[object]*)
- GetFlagged(model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(model*[Model]*, label*[integer]*)
- GetFromIndex(model*[Model]*, index*[integer]*)
- GetMultipleData(component*[constant]*, items*[array]*, options (optional)*[object]*)
- Last(model*[Model]*)
- Pick()
- Select(flag*[Flag]*)
- Total(model*[Model]*)
- UnblankAll(window*[GraphicsWindow]*, model*[Model]*)
- UnblankFlagged(window*[GraphicsWindow]*, model*[Model]*, flag*[Flag]*)
- UnflagAll(model*[Model]*, flag*[Flag]*)

Member functions

- Blank(window*[GraphicsWindow]*)
- Blanked(window*[GraphicsWindow]*)
- ClearFlag(flag*[Flag]*)
- Elements()
- Flagged(flag*[Flag]*)
- GetCompositeData(ipt*[integer]*)
- GetData(component*[constant]*, options (optional)*[object]*)
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Unblank(window*[GraphicsWindow]*)

Part properties

Name	Type	Description
colour	Colour	The colour of the part
composite (read only)	logical	If this is a PART_COMPOSITE. Can be true or false. If there is no ztf file for the model this will be false.
data (read only)	real|array	Component data for a part passed as an argument to GetMultipleData. Note that data will only exist for the instance of the part passed to GetMultipleData. i.e. it is a local property stored on the specific instance. It is not stored in the D3PLOT database
displayMode	integer	The display mode of the part. Can be Constant.WIREFRAME, Constant.HIDDEN, Constant.SHADED or Constant.CURRENT.
elementType (read only)	constant	The type of elements in the part. e.g. Type.SHELL, Type.SOLID etc

include (read only)	integer	The include file number in the model that the part is in
index (read only)	integer	The internal index for the part in D3PLOT (starting at 0)
label (read only)	integer	The Ansys LS-DYNA label for the part
material (read only)	Material	The Material the part has. This is only available if there is a ztf file for the model. If not null will be returned. If this is a PART_COMPOSITE then null will be returned. If the model comes from writing a PTF/d3plot file from PRIMER and the material was latent there, null will be returned. Part.GetCompositeData should be used to get material data in this case
model (read only)	Model	The Model that the part is in
nip (read only)	integer	Number of integration points (layers) present for _COMPOSITE parts
overlayColour	Colour	The overlay colour of the part
overlayMode	integer	The overlay mode of the part. Can be Constant.NO_OVERLAY, Constant.FREE_EDGE, Constant.ALL_EDGES or Constant.CURRENT.
title (read only)	string	The title for the part (or null if no title)
transparency	integer	The transparency of the part between 0 (opaque) and 100 (clear)
type (read only)	constant	The type for the part (will be Type.PART)

Detailed Description

The Part class allows you to inspect parts in a model. See the documentation below for more details.

Details of functions

Blank(window*[GraphicsWindow]*)

Description

Blanks the part in a graphics window

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to blank the part in

Returns

No return value

Example

To blank part p in graphics window g:

```
p.Blank(g);
```

BlankAll(window*[GraphicsWindow]*, model*[Model]*) [static]

Description

Blanks all of the parts in the model

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to blank the parts in

- **model** (Model)

Model that all the parts will be blanked in

Returns

No return value

Example

To blank all of the parts in model m, in graphics window gw:

```
Part.BlankAll(gw, m);
```

BlankFlagged(window*[GraphicsWindow]*, model*[Model]*, flag*[Flag]*) [static]

Description

Blanks all of the parts in the model flagged with a defined flag

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to blank the parts in

- **model** (Model)

Model that the flagged parts will be blanked in

- **flag** (Flag)

Flag (see AllocateFlag) set on the parts to blank

Returns

No return value

Example

To blank all of the parts flagged with flag f in model m, in graphics window gw:

```
Part.BlankFlagged(gw, m, f);
```

Blanked(window*[GraphicsWindow]*)

Description

Checks if the part is blanked in a graphics window or not

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) in which to check if the part is blanked

Returns

true if blanked, false if not

Return type

boolean

Example

To check if part p is blanked in graphics window g:

```
if (p.Blanked(g) ) do_something...
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on a part

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to clear on the part

Returns

No return value

Example

To clear flag f on part p:

```
p.ClearFlag();
```

Elements()

Description

Returns an array containing the elements in the part

Arguments

No arguments

Returns

Array of element objects

Return type

Array

Example

To return the elements for part p:

```
var elements = p.Elements();
```

First(model*[Model]*) [static]

Description

Returns the first part in the model (or null if there are no parts in the model)

Arguments

- **model** (Model)

Model to get first part in

Returns

Part object

Return type

Part

Example

To get the first part in model m:

```
var p = Part.First(m);
```

FlagAll(model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the parts in the model with a defined flag

Arguments

- **model** (Model)

Model that all the parts will be flagged in

- **flag** (Flag)

Flag (see AllocateFlag) to set on the parts

Returns

No return value

Example

To flag all of the parts with flag f in model m:

```
Part.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the part is flagged or not

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to test on the part

Returns

true if flagged, false if not

Return type

boolean

Example

To check if part p has flag f set on it:

```
if (p.Flagged(f) ) do_something...
```

GetAll(model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Part objects or properties for all of the parts in the model. If the optional property argument is not given then an array of Part objects is returned. If the property argument is given, that property value for each part is returned in the array instead of a Part object

Arguments

- **model** (Model)

Model that all the parts are in

- **property (optional)** (string)

Name for property to get for all parts in the model

Returns

Array of Part objects or properties

Return type

Array

Example

To get all of the parts in model m:

```
var p = Part.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'label' for a part) for each part in model m:

```
var a = Part.GetAll(m, 'foo');
```

GetCompositeData(ipt*[integer]*)

Description

Returns the composite data for an integration point in *PART_COMPOSITE.

Arguments

- **ipt** (integer)

The integration point you want the data for. **Note that integration points start at 0, not 1**.

Returns

An array containing the material id and thickness values.

Return type

Array

Example

To get the composite data for the 3rd integration point for part p:

```
if (p.composite && p.nip >= 3)
{
    var ipt_data = p.GetCompositeData(2);
}
```

GetConditionParts(options*[object]*) [static]

Description

Returns an object with all parts in current model filtered into two lists (pass_list and fail_list). Passing parts must have elements that pass the specified condition (indicated by <value> and <mode>) for the specified <component>.

For spotweld data components passing parts must have a beam or solid (hex) spotweld passing the specified condition such that the element of that weld is in the given part.

NOTE: This function only works for scalar components. If <component> is a vector or tensor component, this function will return null.

Arguments

- **options** (object)

Returns an object with all parts in current model filtered into two lists (pass_list and fail_list). Passing parts must have elements that pass the specified condition (indicated by <value> and <mode>) for the specified <component>.

For spotweld data components passing parts must have a beam or solid (hex) spotweld passing the specified condition such that the element of that weld is in the given part.

NOTE: This function only works for scalar components. If <component> is a vector or tensor component, this function will return null.

Object has the following properties:

Name	Type	Description
component	integer	A valid component code (e.g. Component.DX, Component.SXY)
extra (optional)	integer	• The "Extra" solid or shell component id for components Component.SOX or Component.SHX • The ALE mulit-material group id for components Component.AMMG and Component.AMMS • The sub-number for user-defined components Component.UNOS, Component.UNOV, Component.USSS, Component.USST, Component.UBMS, Component.UBMV If any of the above component codes are used, the "extra" argument must be set to a non zero value.
ip (optional)	integer	Through thickness integration point. This can be either 1. A +ve integer that is an integration point id or 2. One of the types Constant.TOP, Constant.MIDDLE or Constant.BOTTOM for shell surfaces only Integration points are only meaningful for some element type / data component combinations: • Shells and Tshells: Stress and strain tensors; Plastic strain; "Extra" data (if written) • Solids: All data components if NINTSLD on the database extent binary card is 8 • Beams: 3 Stresses and 2 strains in non-resultant beam types if BEAMIP on the database extent binary card is > 0 ip and op have become complex data fields, please see the separate section on Defining the Integration point argument. Where the integration point is not relevant this argument may be omitted. If <ip> is not defined, this function will loop through all through thickness integration points for each element to check if the condition is fufilled.
mode	integer	Determines if condition is Constant.LT (<), Constant.LTEQ (<=), Constant.GT (>) or Constant.GTEQ (>=) than <value>.
model	Model object	The model whose parts will be returned
op (optional)	integer	On plan integration point. Defaults to the first one.
value	real	A value to compare element data against.

Returns

Object with the following properties:

Name	Type	Description
failList	array of integers	Failing parts (array of Part objects)
failed	integer	Number of failing parts
passList	array of integers	Passing parts (array of Part objects)
passed	integer	Number of passing parts

Return type

object

Example

```
// Does the filtering for the (scalar) X stress of part elements at integration
point 2 in model 1,
// checking if they are greater than 0.1:
var m = Model.GetFromID(1);
var a = Part.GetConditionParts({model:m, component:Component.SXX, value:0.1,
mode:Constant.GT, ip:2});
for(const part of a.passList)
{
    Message("Part " + part.label + " satisfies the condition.");
}
// Does the filtering for the yield utilisation factor of part elements (looping
through all
// integration points) in model 3, checking if they are greater than or equals
to 1.0:
var m = Model.GetFromID(3);
var b = Part.GetConditionParts({model:m, component:Component.YUTF, value:1.0,
mode:Constant.GTEQ});
```

GetData(component*[constant]*, options (optional)*[object]*)

Description

Returns the value for a data component.
Also see GetMultipleData

Arguments

- **component** (constant)

Component constant to get data for

- **options (optional)** (object)

Object containing options for getting data. Can be any of:

Object has the following properties:

Name	Type	Description
extra	integer	The extra data component number if component Component.SOX for solids, Component.BMX for beams or Component.SHX for shells and thick shells
ip	integer	Integration point number to get the data at (ip >= 1 or one of the constants Constant.TOP, Constant.MIDDLE or Constant.BOTTOM). If the integration point is not defined it will use the integration point defined on the current GUI "data" panel, which defaults to the middle surface for shells, thick shells, and solids, and Mag All for beams, but may vary if changed by an interactive user. If consistent output from a script is required, independent of any prior interactive activity, an explicit integration point or surface should be defined
op	integer	On plane integration point number for shells and thick shells (op >= 1 [default])

referenceFrame	constant	The frame of reference to return values in. Either Constant.GLOBAL (default), Constant.LOCAL, Constant.CYLINDRICAL, Constant.USER_DEFINED or Constant.MATERIAL. This is only necessary for directional components (eg X stress) and then only when something other than the default Constant.GLOBAL coordinate system is to be used
user	integer	The user-defined component number if component Component.UNOS, Component.UNOV, Component.USSS, Component.USST, Component.UBMS or Component.UBMV

Returns

Number if a scalar component, array if a vector or tensor component (or null if the value cannot be calculated because it's not available in the model).
If requesting an invalid component it will throw an error (e.g. Component.AREA of a node).

Return type

real|array

Example

To calculate a component and check it has been calculated (note that in the example, the argument extra is optional):

```
var value = p.GetData(component, {extra: 1});
if (value !== null) do_something...
```

GetFlagged(model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Gets all of the parts in the model flagged with a defined flag. If the optional property argument is not given then an array of Part objects is returned. If the property argument is given, that property value for each part is returned in the array instead of a Part object

Arguments

- **model** (Model)

Model that the flagged parts are in

- **flag** (Flag)

Flag (see AllocateFlag) set on the parts to get

- **property (optional)** (string)

Name for property to get for all flagged parts in the model

Returns

Array of Part objects or properties

Return type

Array

Example

To get all of the parts flagged with flag f in model m:

```
Part.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'label' for a node) for all parts flagged with flag f in model m:

```
var a = Part.GetFlagged(m, f, 'foo');
```

GetFromID(model*[Model]*, label*[integer]*) [static]

Description

Returns the Part object for part in model with label (or null if it does not exist)

Arguments

- **model** (Model)

Model to get part in

- **label** (integer)

The Ansys LS-DYNA label for the part in the model

Returns

Part object

Return type

Part

Example

To get the part in model m with label 1000:

```
var p = Part.GetFromID(m, 1000);
```

GetFromIndex(model*[Model]*, index*[integer]*) [static]

Description

Returns the Part object for part in model with index (or null if it does not exist)

Arguments

- **model** (Model)

Model to get part in

- **index** (integer)

The D3PLOT internal index in the model for part, starting at 0.

Returns

Part object

Return type

Part

Example

To get the 51st part in model m:

```
var p = Part.GetFromIndex(m, 50);
```

GetMultipleData(component*[constant]*, items*[array]*, options (optional)*[object]*) [static]

Description

Returns the value for a data component for multiple parts. For each part a local property called data will be created containing a number if a scalar component, or an array if a vector or tensor component (or null if the value cannot be calculated). The data is also returned as an object.
Also see GetData

Arguments

- **component** (constant)

Component constant to get data for

- **items** (array)

Array of Part objects to get the data for. All of the parts must be in the same model.

- **options (optional)** (object)

Object containing options for getting data. Can be any of:

Object has the following properties:

Name	Type	Description
extra	integer	The extra data component number if component Component.SOX for solids, Component.BMX for beams or Component.SHX for shells and thick shells
ip	integer	Integration point number to get the data at (ip >= 1 or one of the constants Constant.TOP, Constant.MIDDLE or Constant.BOTTOM)
op	integer	On plane integration point number for shells and thick shells (op >= 1 [default])
referenceFrame	constant	The frame of reference to return values in. Either Constant.GLOBAL (default), Constant.LOCAL, Constant.CYLINDRICAL, Constant.USER_DEFINED or Constant.MATERIAL. This is only necessary for directional components (eg X stress) and then only when something other than the default Constant.GLOBAL coordinate system is to be used
user	integer	The user-defined component number if component Component.UNOS, Component.UNOV, Component.USSS, Component.USST, Component.UBMS or Component.UBMV

Returns

Object containing the data. A property is created in the object for each part with the label. The value of the property is a number if a scalar component or an array if a vector or tensor component (or null if the value cannot be calculated)

Return type

object

Example

To calculate a component for parts in array items and use the data property (note that in the example, the argument extra is optional):

```
Part.GetMultipleData(component, items, {extra: 1});
for (i=0; i<items.length; i++)
{
    if (items[i].data !== null) do_something...
}
```

To calculate a component for parts in array items and use the return value (note that in the example, the argument extra is optional):

```
var data = Part.GetMultipleData(component, items, {extra: 1});
for (d in data)
{
    Message("Label is " + d);
    if (data[d] !== null) do_something...
}
```

Last(model[*Model*]) [static]

Description

Returns the last part in the model (or null if there are no parts in the model)

Arguments

- **model** (Model)

Model to get last part in

Returns

Part object

Return type

Part

Example

To get the last part in model m:

```
var p = Part.Last(m);
```

Next()

Description

Returns the next part in the model (or null if there is not one)

Arguments

No arguments

Returns

Part object

Return type

Part

Example

To get the next part after part p:

```
p = p.Next();
```

Pick() [static]

Description

Allows the user to pick a part from the screen

Arguments

No arguments

Returns

Part object or null if cancelled

Return type

Part

Example

To pick a part:

```
var p = Part.Pick();
```

Previous()

Description

Returns the previous part in the model (or null if there is not one)

Arguments

No arguments

Returns

Part object

Return type

Part

Example

To get the previous part before part p:

```
p = p.Previous();
```

Select(flag*[Flag]*) [static]

Description

Selects parts using an object menu

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to use when selecting parts

Returns

The number of parts selected or null if menu cancelled

Return type

integer

Example

To select parts, flagging those selected with flag f:

```
var total = Part.Select(f);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on a part

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to set on the part

Returns

No return value

Example

To set flag f on part p:

```
p.SetFlag(f);
```

Total(model*[Model]*) [static]

Description

Returns the total number of parts in the model

Arguments

- **model** (Model)

Model to get total in

Returns

The number of parts

Return type

integer

Example

To get the number of parts in model m:

```
var total = Part.Total(m);
```

Unblank(window*[GraphicsWindow]*)

Description

Unblanks the part in a graphics window

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to unblank the part in

Returns

No return value

Example

To unblank part p in graphics window g:

```
p.Unblank(g);
```

UnblankAll(window*[GraphicsWindow]*, model*[Model]*) [static]

Description

Unblanks all of the parts in the model

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to unblank the parts in

- **model** (Model)

Model that all the parts will be unblanked in

Returns

No return value

Example

To unblank all of the parts in model m, in graphics window gw:

```
Part.UnblankAll(gw, m);
```

UnblankFlagged(window*[GraphicsWindow]*, model*[Model]*, flag*[Flag]*) [static]

Description

Unblanks all of the parts in the model flagged with a defined flag

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to unblank the parts in

- **model** (Model)

Model that the flagged parts will be unblanked in

- **flag** (Flag)

Flag (see AllocateFlag) set on the parts to unblank

Returns

No return value

Example

To unblank all of the parts flagged with flag f in model m, in graphics window gw:

```
Part.UnblankFlagged(gw, m, f);
```

UnflagAll(model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the parts in the model

Arguments

- **model** (Model)

Model that the defined flag for all parts will be unset in

- **flag** (Flag)

Flag (see AllocateFlag) to unset on the parts

Returns

No return value

Example

To unset flag f on all of the parts in model m:

```
Part.UnflagAll(m, f);
```

PopupWindow class

The PopupWindow class allows you to create popup windows for a graphical user interface. More...

The D3PLOT JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Member functions

- Hide()

PopupWindow properties

Name	Type	Description
persistent	boolean	If the popup window will remain mapped when a button is pressed in it. By default (false) when a button is pressed in the popup window the popup will be unmapped. If set to true then the popup will remain mapped until the user clicks out of the window or hides it by calling Hide()

Detailed Description

The PopupWindow class allows you to make popup windows (that you can place Widgets in) and link them to Widgets. The popup window is then displayed by right clicking on the Widget the popup is linked to. The following very simple example shows how to create a popup window and link it to a label Widget.

```
// Create popup window
var pw = new PopupWindow();
// Create some widgets in the popup window
var pl = new Widget(pw, Widget.LABEL,   1, 30,  1,  7, "Label");
var pb = new Widget(pw, Widget.BUTTON,  1, 30,  7, 13, "Button");
var pt = new Widget(pw, Widget.TEXTBOX, 1, 30, 20, 26, "Textbox");
// Create window with title "Popup example" from 0.8-1.0 in x and 0.5-0.6 in y
var w = new Window("Popup example", 0.8, 1.0, 0.5, 0.6);
// Create label widget
var l = new Widget(w, Widget.LABEL,   1, 50, 1,  7, "Right click for popup...");
// link popup window to widget
l.popupWindow = pw;
// Assign the onPopup callback method to the function 'do_popup'
// This is only required if you want to make any changes before the popup
appears
l.onPopup  = do_popup;
// Show the widget and start event loop
w.Show();
////////////////////////////////////////////////////////////////////////////
function do_popup()
{
    Message("Showing popup");
}
```

See the documentation below and the Widget class for more details.

Constructor

new PopupWindow()

Description

Create a new PopupWindow object.

Arguments

No arguments

Returns

PopupWindow object

Return type

PopupWindow

Example

To create a PopupWindow containing the buttons "Create" and "Edit" and link it to button b:

```
var pw = new PopupWindow();
var c = new Widget(pw, Widget.BUTTON, 1, 30, 1, 7,  "Create");
var e = new Widget(pw, Widget.BUTTON, 1, 30, 7, 13, "Edit");
b.popupWindow = pw;
```

Details of functions

Hide()

Description

Hides (unmaps) the popup window.

Arguments

No arguments

Returns

No return value

Example

To hide popup window w:

```
w.Hide();
```

Segment class

The Segment class gives you access to contact segments in D3PLOT. More...

The D3PLOT JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(window*[GraphicsWindow]*, model*[Model]*)
- BlankFlagged(window*[GraphicsWindow]*, model*[Model]*, flag*[Flag]*)
- First(model*[Model]*)
- FlagAll(model*[Model]*, flag*[Flag]*)
- GetAll(model*[Model]*, property (optional)*[string]*)
- GetFlagged(model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(model*[Model]*, label*[integer]*)
- GetFromIndex(model*[Model]*, index*[integer]*)
- GetMultipleData(component*[constant]*, items*[array]*, options (optional)*[object]*)
- Last(model*[Model]*)
- Pick()
- Select(flag*[Flag]*)
- Total(model*[Model]*)
- UnblankAll(window*[GraphicsWindow]*, model*[Model]*)
- UnblankFlagged(window*[GraphicsWindow]*, model*[Model]*, flag*[Flag]*)
- UnflagAll(model*[Model]*, flag*[Flag]*)

Member functions

- Blank(window*[GraphicsWindow]*)
- Blanked(window*[GraphicsWindow]*)
- ClearFlag(flag*[Flag]*)
- Flagged(flag*[Flag]*)
- GetData(component*[constant]*, options (optional)*[object]*)
- LocalAxes()
- Next()
- Previous()
- SetFlag(flag*[Flag]*)
- Topology()
- Unblank(window*[GraphicsWindow]*)

Segment properties

Name	Type	Description
colour	Colour	The colour of the segment
data (read only)	real|array	Component data for a segment passed as an argument to GetMultipleData. Note that data will only exist for the instance of the segment passed to GetMultipleData. i.e. it is a local property stored on the specific instance. It is not stored in the D3PLOT database
displayMode	integer	The display mode of the segment. Can be Constant.WIREFRAME, Constant.HIDDEN, Constant.SHADED or Constant.CURRENT.
include (read only)	integer	The include file number in the model that the segment is in
index (read only)	integer	The internal index for the segment in D3PLOT (starting at 0)

label (read only)	integer	The Ansys LS-DYNA label for the segment
material (read only)	Material	The Material the segment has. This is only available if there is a ztf file for the model. If not null will be returned. If this is a PART_COMPOSITE then null will be returned. If the model comes from writing a PTF/d3plot file from PRIMER and the material was latent there, null will be returned. Part.GetCompositeData should be used to get material data in this case
model (read only)	Model	The Model that the segment is in
overlayColour	Colour	The overlay colour of the segment
overlayMode	integer	The overlay mode of the segment. Can be Constant.NO_OVERLAY, Constant.FREE_EDGE, Constant.ALL_EDGES or Constant.CURRENT.
part (read only)	Part	The Part the segment is in
transparency	integer	The transparency of the segment between 0 (opaque) and 100 (clear)
type (read only)	constant	The type for the segment (will be Type.SEGMENT)

Detailed Description

The Segment class allows you to inspect contact segments in a model. See the documentation below for more details.

Details of functions

Blank(window*[GraphicsWindow]*)

Description

Blanks the segment in a graphics window

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to blank the segment in

Returns

No return value

Example

To blank segment s in graphics window g:

```
s.Blank(g);
```

BlankAll(window*[GraphicsWindow]*, model*[Model]*) [static]

Description

Blanks all of the segments in the model

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to blank the segments in

- **model** (Model)

Model that all the segments will be blanked in

Returns

No return value

Example

To blank all of the segments in model m, in graphics window gw:

```
Segment.BlankAll(gw, m);
```

BlankFlagged(window*[GraphicsWindow]*, model*[Model]*, flag*[Flag]*) [static]

Description

Blanks all of the segments in the model flagged with a defined flag

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to blank the segments in

- **model** (Model)

Model that the flagged segments will be blanked in

- **flag** (Flag)

Flag (see AllocateFlag) set on the segments to blank

Returns

No return value

Example

To blank all of the segments flagged with flag f in model m, in graphics window gw:

```
Segment.BlankFlagged(gw, m, f);
```

Blanked(window*[GraphicsWindow]*)

Description

Checks if the segment is blanked in a graphics window or not

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) in which to check if the segment is blanked

Returns

true if blanked, false if not

Return type

boolean

Example

To check if segment s is blanked in graphics window g:

```
if (s.Blanked(g) ) do_something...
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on a segment

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to clear on the segment

Returns

No return value

Example

To clear flag f on segment s:

```
s.ClearFlag();
```

First(model*[Model]*) [static]

Description

Returns the first segment in the model (or null if there are no segments in the model)

Arguments

- **model** (Model)

Model to get first segment in

Returns

Segment object

Return type

Segment

Example

To get the first segment in model m:

```
var s = Segment.First(m);
```

FlagAll(model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the segments in the model with a defined flag

Arguments

- **model** (Model)

Model that all the segments will be flagged in

- **flag** (Flag)

Flag (see AllocateFlag) to set on the segments

Returns

No return value

Example

To flag all of the segments with flag f in model m:

```
Segment.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the segment is flagged or not

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to test on the segment

Returns

true if flagged, false if not

Return type

boolean

Example

To check if segment s has flag f set on it:

```
if (s.Flagged(f) ) do_something...
```

GetAll(model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Segment objects or properties for all of the segments in the model. If the optional property argument is not given then an array of Segment objects is returned. If the property argument is given, that property value for each segment is returned in the array instead of a Segment object

Arguments

- **model** (Model)

Model that all the segments are in

- **property (optional)** (string)

Name for property to get for all segments in the model

Returns

Array of Segment objects or properties

Return type

Array

Example

To get all of the segments in model m:

```
var s = Segment.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'label' for a segment) for each segment in model m:

```
var a = Segment.GetAll(m, 'foo');
```

GetData(component*[constant]*, options (optional)*[object]*)

Description

Returns the value for a data component.
Also see GetMultipleData

Arguments

- **component** (constant)

Component constant to get data for

- **options (optional)** (object)

Object containing options for getting data. Can be any of:

Object has the following properties:

Name	Type	Description
extra	integer	The extra data component number if component Component.SOX for solids, Component.BMX for beams or Component.SHX for shells and thick shells
ip	integer	Integration point number to get the data at (ip >= 1 or one of the constants Constant.TOP, Constant.MIDDLE or Constant.BOTTOM). If the integration point is not defined it will use the integration point defined on the current GUI "data" panel, which defaults to the middle surface for shells, thick shells, and solids, and Mag All for beams, but may vary if changed by an interactive user. If consistent output from a script is required, independent of any prior interactive activity, an explicit integration point or surface should be defined
op	integer	On plane integration point number for shells and thick shells (op >= 1 [default])
referenceFrame	constant	The frame of reference to return values in. Either Constant.GLOBAL (default), Constant.LOCAL, Constant.CYLINDRICAL, Constant.USER_DEFINED or Constant.MATERIAL. This is only necessary for directional components (eg X stress) and then only when something other than the default Constant.GLOBAL coordinate system is to be used
user	integer	The user-defined component number if component Component.UNOS, Component.UNOV, Component.USSS, Component.USST, Component.UBMS or Component.UBMV

Returns

Number if a scalar component, array if a vector or tensor component (or null if the value cannot be calculated because it’s not available in the model).
If requesting an invalid component it will throw an error (e.g. Component.AREA of a node).

Return type

real|array

Example

To calculate a component and check it has been calculated (note that in the example, the argument extra is optional):

```
var value = s.GetData(component, {extra: 1});
if (value !== null) do_something...
```

GetFlagged(model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Gets all of the segments in the model flagged with a defined flag. If the optional property argument is not given then an array of Segment objects is returned. If the property argument is given, that property value for each segment is returned in the array instead of a Segment object

Arguments

- **model** (Model)

Model that the flagged segments are in

- **flag** (Flag)

Flag (see AllocateFlag) set on the segments to get

- **property (optional)** (string)

Name for property to get for all flagged segments in the model

Returns

Array of Segment objects or properties

Return type

Array

Example

To get all of the segments flagged with flag f in model m:

```
Segment.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'label' for a node) for all segments flagged with flag f in model m:

```
var a = Segment.GetFlagged(m, f, 'foo');
```

GetFromID(model*[Model]*, label*[integer]*) [static]

Description

Returns the Segment object for segment in model with label (or null if it does not exist)

Arguments

- **model** (Model)

Model to get segment in

- **label** (integer)

The Ansys LS-DYNA label for the segment in the model

Returns

Segment object

Return type

Segment

Example

To get the segment in model m with label 1000:

```
var s = Segment.GetFromID(m, 1000);
```

GetFromIndex(model*[Model]*, index*[integer]*) [static]

Description

Returns the Segment object for segment in model with index (or null if it does not exist)

Arguments

- **model** (Model)

Model to get segment in

- **index** (integer)

The D3PLOT internal index in the model for segment, starting at 0.

Returns

Segment object

Return type

Segment

Example

To get the 51st segment in model m:

```
var s = Segment.GetFromIndex(m, 50);
```

GetMultipleData(component*[constant]*, items*[array]*, options (optional)*[object]*) [static]

Description

Returns the value for a data component for multiple segments. For each segment a local property called data will be created containing a number if a scalar component, or an array if a vector or tensor component (or null if the value cannot be calculated). The data is also returned as an object.
Also see GetData

Arguments

- **component** (constant)

Component constant to get data for

- **items** (array)

Array of Segment objects to get the data for. All of the segments must be in the same model.

- **options (optional)** (object)

Object containing options for getting data. Can be any of:

Object has the following properties:

Name	Type	Description
extra	integer	The extra data component number if component Component.SOX for solids, Component.BMX for beams or Component.SHX for shells and thick shells
ip	integer	Integration point number to get the data at (ip >= 1 or one of the constants Constant.TOP, Constant.MIDDLE or Constant.BOTTOM)
op	integer	On plane integration point number for shells and thick shells (op >= 1 [default])
referenceFrame	constant	The frame of reference to return values in. Either Constant.GLOBAL (default), Constant.LOCAL, Constant.CYLINDRICAL, Constant.USER_DEFINED or Constant.MATERIAL. This is only necessary for directional components (eg X stress) and then only when something other than the default Constant.GLOBAL coordinate system is to be used
user	integer	The user-defined component number if component Component.UNOS, Component.UNOV, Component.USSS, Component.USST, Component.UBMS or Component.UBMV

Returns

Object containing the data. A property is created in the object for each segment with the label. The value of the property is a number if a scalar component or an array if a vector or tensor component (or null if the value cannot be calculated)

Return type

object

Example

To calculate a component for segments in array items and use the data property (note that in the example, the argument extra is optional):

```
Segment.GetMultipleData(component, items, {extra: 1});
for (i=0; i<items.length; i++)
{
    if (items[i].data !== null) do_something...
}
```

To calculate a component for segments in array items and use the return value (note that in the example, the argument extra is optional):

```
var data = Segment.GetMultipleData(component, items, {extra: 1});
for (d in data)
{
    Message("Label is " + d);
    if (data[d] !== null) do_something...
}
```

Last(model[*Model*]) [static]

Description

Returns the last segment in the model (or null if there are no segments in the model)

Arguments

- **model** (Model)

Model to get last segment in

Returns

Segment object

Return type

Segment

Example

To get the last segment in model m:

```
var s = Segment.Last(m);
```

LocalAxes()

Description

Returns the local axes of the element in model space, expressed as direction cosines in a 2D array. Beam elements must have 3 nodes to be able to return local axes.

Arguments

No arguments

Returns

array of arrays

Return type

Array

Example

To get the local axes for segment s:

```
var axes = s.LocalAxes();
var xAxis = [ axes[0][0], axes[0][1], axes[0][2] ];
var yAxis = [ axes[1][0], axes[1][1], axes[1][2] ];
var zAxis = [ axes[2][0], axes[2][1], axes[2][2] ];
```

Next()

Description

Returns the next segment in the model (or null if there is not one)

Arguments

No arguments

Returns

Segment object

Return type

Segment

Example

To get the next segment after segment s:

```
s = s.Next();
```

Pick() [static]

Description

Allows the user to pick a segment from the screen

Arguments

No arguments

Returns

Segment object or null if cancelled

Return type

Segment

Example

To pick a segment:

```
var s = Segment.Pick();
```

Previous()

Description

Returns the previous segment in the model (or null if there is not one)

Arguments

No arguments

Returns

Segment object

Return type

Segment

Example

To get the previous segment before segment s:

```
s = s.Previous();
```

Select(flag*[Flag]*) [static]

Description

Selects segments using an object menu

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to use when selecting segments

Returns

The number of segments selected or null if menu cancelled

Return type

integer

Example

To select segments, flagging those selected with flag f:

```
var total = Segment.Select(f);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on a segment

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to set on the segment

Returns

No return value

Example

To set flag f on segment s:

```
s.SetFlag(f);
```

Topology()

Description

Returns the topology for the segment in the model

Arguments

No arguments

Returns

array of Node objects

Return type

Array

Example

To get the topology for segment s:

```
var topology = s.Topology();
```

Total(model*[Model]*) [static]

Description

Returns the total number of segments in the model

Arguments

- **model** (Model)

Model to get total in

Returns

The number of segments

Return type

integer

Example

To get the number of segments in model m:

```
var total = Segment.Total(m);
```

Unblank(window*[GraphicsWindow]*)

Description

Unblanks the segment in a graphics window

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to unblank the segment in

Returns

No return value

Example

To unblank segment s in graphics window g:

```
s.Unblank(g);
```

UnblankAll(window*[GraphicsWindow]*, model*[Model]*) [static]

Description

Unblanks all of the segments in the model

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to unblank the segments in

- **model** (Model)

Model that all the segments will be unblanked in

Returns

No return value

Example

To unblank all of the segments in model m, in graphics window gw:

```
Segment.UnblankAll(gw, m);
```

UnblankFlagged(window*[GraphicsWindow]*, model*[Model]*, flag*[Flag]*) [static]

Description

Unblanks all of the segments in the model flagged with a defined flag

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to unblank the segments in

- **model** (Model)

Model that the flagged segments will be unblanked in

- **flag** (Flag)

Flag (see AllocateFlag) set on the segments to unblank

Returns

No return value

Example

To unblank all of the segments flagged with flag f in model m, in graphics window gw:

```
Segment.UnblankFlagged(gw, m, f);
```

UnflagAll(model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the segments in the model

Arguments

- **model** (Model)

Model that the defined flag for all segments will be unset in

- **flag** (Flag)

Flag (see AllocateFlag) to unset on the segments

Returns

No return value

Example

To unset flag f on all of the segments in model m:

```
Segment.UnflagAll(m, f);
```

SetBeam class

The SetBeam class gives you access to beam sets in D3PLOT. More...

The D3PLOT JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- First(model*[Model]*)
- FlagAll(model*[Model]*, flag*[Flag]*)
- GetAll(model*[Model]*, property (optional)*[string]*)
- GetFlagged(model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(model*[Model]*, label*[integer]*)
- GetFromIndex(model*[Model]*, index*[integer]*)
- Last(model*[Model]*)
- Total(model*[Model]*)
- UnflagAll(model*[Model]*, flag*[Flag]*)

Member functions

- AllItems()
- ClearFlag(flag*[Flag]*)
- Flagged(flag*[Flag]*)
- Item(index*[integer]*)
- Next()
- Previous()
- SetFlag(flag*[Flag]*)

SetBeam properties

Name	Type	Description
include (read only)	integer	The include file number in the model that the beam set is in
index (read only)	integer	The internal index for the beam set in D3PLOT (starting at 0)
label (read only)	integer	The Ansys LS-DYNA label for the beam set
model (read only)	Model	The Model that the beam set is in
title (read only)	string	The title for the beam set (or null if no title). This is only available if there is a ztf file for the model. If not null will be returned.
total (read only)	integer	The total number of beam items in the beam set
type (read only)	constant	The type for the beam set (will be Type.SET_BEAM)

Detailed Description

The SetBeam class allows you to view sets in D3PLOT. There are various methods and properties available. This class requires a ztf file to be present for the model. See the documentation below for more details.

Details of functions

AllItems()

Description

Returns all of the beam items for the beam set in the model

Arguments

No arguments

Returns

array of Beam objects

Return type

Array

Example

To get the beam items in beam set s:

```
var items = s.AllItems();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on a beam set

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to clear on the beam set

Returns

No return value

Example

To clear flag f on beam set s:

```
s.ClearFlag();
```

First(model*[Model]*) [static]

Description

Returns the first beam set in the model (or null if there are no beam sets in the model)

Arguments

- **model** (Model)

Model to get first beam set in

Returns

SetBeam object

Return type

SetBeam

Example

To get the first beam set in model m:

```
var s = SetBeam.First(m);
```

FlagAll(model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the beam sets in the model with a defined flag

Arguments

- **model** (Model)

Model that all the beam sets will be flagged in

- **flag** (Flag)

Flag (see AllocateFlag) to set on the beam sets

Returns

No return value

Example

To flag all of the beam sets with flag f in model m:

```
SetBeam.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the beam set is flagged or not

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to test on the beam set

Returns

true if flagged, false if not

Return type

boolean

Example

To check if beam set s has flag f set on it:

```
if (s.Flagged(f) ) do_something...
```

GetAll(model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of SetBeam objects or properties for all of the beam sets in the model. If the optional property argument is not given then an array of SetBeam objects is returned. If the property argument is given, that property value for each beam set is returned in the array instead of a SetBeam object

Arguments

- **model** (Model)

Model that all the beam sets are in

- **property (optional)** (string)

Name for property to get for all beam sets in the model

Returns

Array of SetBeam objects or properties

Return type

Array

Example

To get all of the beam sets in model m:

```
var s = SetBeam.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'label' for a beam set) for each beam set in model m:

```
var a = SetBeam.GetAll(m, 'foo');
```

GetFlagged(model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Gets all of the beam sets in the model flagged with a defined flag. If the optional property argument is not given then an array of SetBeam objects is returned. If the property argument is given, that property value for each beam set is returned in the array instead of a SetBeam object

Arguments

- **model** (Model)

Model that the flagged beam sets are in

- **flag** (Flag)

Flag (see AllocateFlag) set on the beam sets to get

- **property (optional)** (string)

Name for property to get for all flagged beam sets in the model

Returns

Array of SetBeam objects or properties

Return type

Array

Example

To get all of the beam sets flagged with flag f in model m:

```
SetBeam.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'label' for a node) for all beam sets flagged with flag f in model m:

```
var a = SetBeam.GetFlagged(m, f, 'foo');
```

GetFromID(model*[Model]*, label*[integer]*) [static]

Description

Returns the SetBeam object for beam set in model with label (or null if it does not exist)

Arguments

- **model** (Model)

Model to get beam set in

- **label** (integer)

The Ansys LS-DYNA label for the beam set in the model

Returns

SetBeam object

Return type

SetBeam

Example

To get the beam set in model m with label 1000:

```
var s = SetBeam.GetFromID(m, 1000);
```

GetFromIndex(model*[Model]*, index*[integer]*) [static]

Description

Returns the SetBeam object for beam set in model with index (or null if it does not exist)

Arguments

- **model** (Model)

Model to get beam set in

- **index** (integer)

The D3PLOT internal index in the model for beam set, starting at 0.

Returns

SetBeam object

Return type

SetBeam

Example

To get the 51st beam set in model m:

```
var s = SetBeam.GetFromIndex(m, 50);
```

Item(index*[integer]*)

Description

Returns a beam item from the beam set in the model

Arguments

- **index** (integer)

The index in the beam set to get the beam from (0 <= index < total)

Returns

Beam object

Return type

Beam

Example

To get the 10th beam in beam set s:

```
var items = s.Item(9);
```

Last(model*[Model]*) [static]

Description

Returns the last beam set in the model (or null if there are no beam sets in the model)

Arguments

- **model** (Model)

Model to get last beam set in

Returns

SetBeam object

Return type

SetBeam

Example

To get the last beam set in model m:

```
var s = SetBeam.Last(m);
```

Next()

Description

Returns the next beam set in the model (or null if there is not one)

Arguments

No arguments

Returns

SetBeam object

Return type

SetBeam

Example

To get the next beam set after beam set s:

```
s = s.Next();
```

Previous()

Description

Returns the previous beam set in the model (or null if there is not one)

Arguments

No arguments

Returns

SetBeam object

Return type

SetBeam

Example

To get the previous beam set before beam set s:

```
s = s.Previous();
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on a beam set

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to set on the beam set

Returns

No return value

Example

To set flag f on beam set s:

```
s.SetFlag(f);
```

Total(model*[Model]*) [static]

Description

Returns the total number of beam sets in the model

Arguments

- **model** (Model)

Model to get total in

Returns

The number of beam sets

Return type

integer

Example

To get the number of beam sets in model m:

```
var total = SetBeam.Total(m);
```

UnflagAll(model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the beam sets in the model

Arguments

- **model** (Model)

Model that the defined flag for all beam sets will be unset in

- **flag** (Flag)

Flag (see AllocateFlag) to unset on the beam sets

Returns

No return value

Example

To unset flag f on all of the beam sets in model m:

```
SetBeam.UnflagAll(m, f);
```

SetNode class

The SetNode class gives you access to node sets in D3PLOT. More...

The D3PLOT JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- First(model*[Model]*)
- FlagAll(model*[Model]*, flag*[Flag]*)
- GetAll(model*[Model]*, property (optional)*[string]*)
- GetFlagged(model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(model*[Model]*, label*[integer]*)
- GetFromIndex(model*[Model]*, index*[integer]*)
- Last(model*[Model]*)
- Total(model*[Model]*)
- UnflagAll(model*[Model]*, flag*[Flag]*)

Member functions

- AllItems()
- ClearFlag(flag*[Flag]*)
- Flagged(flag*[Flag]*)
- Item(index*[integer]*)
- Next()
- Previous()
- SetFlag(flag*[Flag]*)

SetNode properties

Name	Type	Description
include (read only)	integer	The include file number in the model that the node set is in
index (read only)	integer	The internal index for the node set in D3PLOT (starting at 0)
label (read only)	integer	The Ansys LS-DYNA label for the node set
model (read only)	Model	The Model that the node set is in
title (read only)	string	The title for the node set (or null if no title). This is only available if there is a ztf file for the model. If not null will be returned.
total (read only)	integer	The total number of node items in the node set
type (read only)	constant	The type for the node set (will be Type.SET_NODE)

Detailed Description

The SetNode class allows you to view sets in D3PLOT. There are various methods and properties available. This class requires a ztf file to be present for the model. See the documentation below for more details.

Details of functions

AllItems()

Description

Returns all of the node items for the node set in the model

Arguments

No arguments

Returns

array of Node objects

Return type

Array

Example

To get the node items in node set s:

```
var items = s.AllItems();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on a node set

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to clear on the node set

Returns

No return value

Example

To clear flag f on node set s:

```
s.ClearFlag();
```

First(model*[Model]*) [static]

Description

Returns the first node set in the model (or null if there are no node sets in the model)

Arguments

- **model** (Model)

Model to get first node set in

Returns

SetNode object

Return type

SetNode

Example

To get the first node set in model m:

```
var s = SetNode.First(m);
```

FlagAll(model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the node sets in the model with a defined flag

Arguments

- **model** (Model)

Model that all the node sets will be flagged in

- **flag** (Flag)

Flag (see AllocateFlag) to set on the node sets

Returns

No return value

Example

To flag all of the node sets with flag f in model m:

```
SetNode.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the node set is flagged or not

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to test on the node set

Returns

true if flagged, false if not

Return type

boolean

Example

To check if node set s has flag f set on it:

```
if (s.Flagged(f) ) do_something...
```

GetAll(model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of SetNode objects or properties for all of the node sets in the model. If the optional property argument is not given then an array of SetNode objects is returned. If the property argument is given, that property value for each node set is returned in the array instead of a SetNode object

Arguments

- **model** (Model)

Model that all the node sets are in

- **property (optional)** (string)

Name for property to get for all node sets in the model

Returns

Array of SetNode objects or properties

Return type

Array

Example

To get all of the node sets in model m:

```
var s = SetNode.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'label' for a node set) for each node set in model m:

```
var a = SetNode.GetAll(m, 'foo');
```

GetFlagged(model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Gets all of the node sets in the model flagged with a defined flag. If the optional property argument is not given then an array of SetNode objects is returned. If the property argument is given, that property value for each node set is returned in the array instead of a SetNode object

Arguments

- **model** (Model)

Model that the flagged node sets are in

- **flag** (Flag)

Flag (see AllocateFlag) set on the node sets to get

- **property (optional)** (string)

Name for property to get for all flagged node sets in the model

Returns

Array of SetNode objects or properties

Return type

Array

Example

To get all of the node sets flagged with flag f in model m:

```
SetNode.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'label' for a node) for all node sets flagged with flag f in model m:

```
var a = SetNode.GetFlagged(m, f, 'foo');
```

GetFromID(model*[Model]*, label*[integer]*) [static]

Description

Returns the SetNode object for node set in model with label (or null if it does not exist)

Arguments

- **model** (Model)

Model to get node set in

- **label** (integer)

The Ansys LS-DYNA label for the node set in the model

Returns

SetNode object

Return type

SetNode

Example

To get the node set in model m with label 1000:

```
var s = SetNode.GetFromID(m, 1000);
```

GetFromIndex(model*[Model]*, index*[integer]*) [static]

Description

Returns the SetNode object for node set in model with index (or null if it does not exist)

Arguments

- **model** (Model)

Model to get node set in

- **index** (integer)

The D3PLOT internal index in the model for node set, starting at 0.

Returns

SetNode object

Return type

SetNode

Example

To get the 51st node set in model m:

```
var s = SetNode.GetFromIndex(m, 50);
```

Item(index*[integer]*)

Description

Returns a node item from the node set in the model

Arguments

- **index** (integer)

The index in the node set to get the node from (0 <= index < total)

Returns

Node object

Return type

Node

Example

To get the 10th node in node set s:

```
var items = s.Item(9);
```

Last(model*[Model]*) [static]

Description

Returns the last node set in the model (or null if there are no node sets in the model)

Arguments

- **model** (Model)

Model to get last node set in

Returns

SetNode object

Return type

SetNode

Example

To get the last node set in model m:

```
var s = SetNode.Last(m);
```

Next()

Description

Returns the next node set in the model (or null if there is not one)

Arguments

No arguments

Returns

SetNode object

Return type

SetNode

Example

To get the next node set after node set s:

```
s = s.Next();
```

Previous()

Description

Returns the previous node set in the model (or null if there is not one)

Arguments

No arguments

Returns

SetNode object

Return type

SetNode

Example

To get the previous node set before node set s:

```
s = s.Previous();
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on a node set

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to set on the node set

Returns

No return value

Example

To set flag f on node set s:

```
s.SetFlag(f);
```

Total(model*[Model]*) [static]

Description

Returns the total number of node sets in the model

Arguments

- **model** (Model)

Model to get total in

Returns

The number of node sets

Return type

integer

Example

To get the number of node sets in model m:

```
var total = SetNode.Total(m);
```

UnflagAll(model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the node sets in the model

Arguments

- **model** (Model)

Model that the defined flag for all node sets will be unset in

- **flag** (Flag)

Flag (see AllocateFlag) to unset on the node sets

Returns

No return value

Example

To unset flag f on all of the node sets in model m:

```
SetNode.UnflagAll(m, f);
```

SetPart class

The SetPart class gives you access to part sets in D3PLOT. More...

The D3PLOT JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- First(model*[Model]*)
- FlagAll(model*[Model]*, flag*[Flag]*)
- GetAll(model*[Model]*, property (optional)*[string]*)
- GetFlagged(model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(model*[Model]*, label*[integer]*)
- GetFromIndex(model*[Model]*, index*[integer]*)
- Last(model*[Model]*)
- Total(model*[Model]*)
- UnflagAll(model*[Model]*, flag*[Flag]*)

Member functions

- AllItems()
- ClearFlag(flag*[Flag]*)
- Flagged(flag*[Flag]*)
- Item(index*[integer]*)
- Next()
- Previous()
- SetFlag(flag*[Flag]*)

SetPart properties

Name	Type	Description
include (read only)	integer	The include file number in the model that the part set is in
index (read only)	integer	The internal index for the part set in D3PLOT (starting at 0)
label (read only)	integer	The Ansys LS-DYNA label for the part set
model (read only)	Model	The Model that the part set is in
title (read only)	string	The title for the part set (or null if no title). This is only available if there is a ztf file for the model. If not null will be returned.
total (read only)	integer	The total number of part items in the part set
type (read only)	constant	The type for the part set (will be Type.SET_PART)

Detailed Description

The SetPart class allows you to view sets in D3PLOT. There are various methods and properties available. This class requires a ztf file to be present for the model. See the documentation below for more details.

Details of functions

AllItems()

Description

Returns all of the part items for the part set in the model

Arguments

No arguments

Returns

array of Part objects

Return type

Array

Example

To get the part items in part set s:

```
var items = s.AllItems();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on a part set

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to clear on the part set

Returns

No return value

Example

To clear flag f on part set s:

```
s.ClearFlag();
```

First(model*[Model]*) [static]

Description

Returns the first part set in the model (or null if there are no part sets in the model)

Arguments

- **model** (Model)

Model to get first part set in

Returns

SetPart object

Return type

SetPart

Example

To get the first part set in model m:

```
var s = SetPart.First(m);
```

FlagAll(model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the part sets in the model with a defined flag

Arguments

- **model** (Model)

Model that all the part sets will be flagged in

- **flag** (Flag)

Flag (see AllocateFlag) to set on the part sets

Returns

No return value

Example

To flag all of the part sets with flag f in model m:

```
SetPart.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the part set is flagged or not

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to test on the part set

Returns

true if flagged, false if not

Return type

boolean

Example

To check if part set s has flag f set on it:

```
if (s.Flagged(f) ) do_something...
```

GetAll(model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of SetPart objects or properties for all of the part sets in the model. If the optional property argument is not given then an array of SetPart objects is returned. If the property argument is given, that property value for each part set is returned in the array instead of a SetPart object

Arguments

- **model** (Model)

Model that all the part sets are in

- **property (optional)** (string)

Name for property to get for all part sets in the model

Returns

Array of SetPart objects or properties

Return type

Array

Example

To get all of the part sets in model m:

```
var s = SetPart.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'label' for a part set) for each part set in model m:

```
var a = SetPart.GetAll(m, 'foo');
```

GetFlagged(model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Gets all of the part sets in the model flagged with a defined flag. If the optional property argument is not given then an array of SetPart objects is returned. If the property argument is given, that property value for each part set is returned in the array instead of a SetPart object

Arguments

- **model** (Model)

Model that the flagged part sets are in

- **flag** (Flag)

Flag (see AllocateFlag) set on the part sets to get

- **property (optional)** (string)

Name for property to get for all flagged part sets in the model

Returns

Array of SetPart objects or properties

Return type

Array

Example

To get all of the part sets flagged with flag f in model m:

```
SetPart.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'label' for a node) for all part sets flagged with flag f in model m:

```
var a = SetPart.GetFlagged(m, f, 'foo');
```

GetFromID(model*[Model]*, label*[integer]*) [static]

Description

Returns the SetPart object for part set in model with label (or null if it does not exist)

Arguments

- **model** (Model)

Model to get part set in

- **label** (integer)

The Ansys LS-DYNA label for the part set in the model

Returns

SetPart object

Return type

SetPart

Example

To get the part set in model m with label 1000:

```
var s = SetPart.GetFromID(m, 1000);
```

GetFromIndex(model*[Model]*, index*[integer]*) [static]

Description

Returns the SetPart object for part set in model with index (or null if it does not exist)

Arguments

- **model** (Model)

Model to get part set in

- **index** (integer)

The D3PLOT internal index in the model for part set, starting at 0.

Returns

SetPart object

Return type

SetPart

Example

To get the 51st part set in model m:

```
var s = SetPart.GetFromIndex(m, 50);
```

Item(index*[integer]*)

Description

Returns a part item from the part set in the model

Arguments

- **index** (integer)

The index in the part set to get the part from (0 <= index < total)

Returns

Part object

Return type

Part

Example

To get the 10th part in part set s:

```
var items = s.Item(9);
```

Last(model*[Model]*) [static]

Description

Returns the last part set in the model (or null if there are no part sets in the model)

Arguments

- **model** (Model)

Model to get last part set in

Returns

SetPart object

Return type

SetPart

Example

To get the last part set in model m:

```
var s = SetPart.Last(m);
```

Next()

Description

Returns the next part set in the model (or null if there is not one)

Arguments

No arguments

Returns

SetPart object

Return type

SetPart

Example

To get the next part set after part set s:

```
s = s.Next();
```

Previous()

Description

Returns the previous part set in the model (or null if there is not one)

Arguments

No arguments

Returns

SetPart object

Return type

SetPart

Example

To get the previous part set before part set s:

```
s = s.Previous();
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on a part set

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to set on the part set

Returns

No return value

Example

To set flag f on part set s:

```
s.SetFlag(f);
```

Total(model*[Model]*) [static]

Description

Returns the total number of part sets in the model

Arguments

- **model** (Model)

Model to get total in

Returns

The number of part sets

Return type

integer

Example

To get the number of part sets in model m:

```
var total = SetPart.Total(m);
```

UnflagAll(model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the part sets in the model

Arguments

- **model** (Model)

Model that the defined flag for all part sets will be unset in

- **flag** (Flag)

Flag (see AllocateFlag) to unset on the part sets

Returns

No return value

Example

To unset flag f on all of the part sets in model m:

```
SetPart.UnflagAll(m, f);
```

SetShell class

The SetShell class gives you access to shell sets in D3PLOT. More...

The D3PLOT JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- First(model*[Model]*)
- FlagAll(model*[Model]*, flag*[Flag]*)
- GetAll(model*[Model]*, property (optional)*[string]*)
- GetFlagged(model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(model*[Model]*, label*[integer]*)
- GetFromIndex(model*[Model]*, index*[integer]*)
- Last(model*[Model]*)
- Total(model*[Model]*)
- UnflagAll(model*[Model]*, flag*[Flag]*)

Member functions

- AllItems()
- ClearFlag(flag*[Flag]*)
- Flagged(flag*[Flag]*)
- Item(index*[integer]*)
- Next()
- Previous()
- SetFlag(flag*[Flag]*)

SetShell properties

Name	Type	Description
include (read only)	integer	The include file number in the model that the shell set is in
index (read only)	integer	The internal index for the shell set in D3PLOT (starting at 0)
label (read only)	integer	The Ansys LS-DYNA label for the shell set
model (read only)	Model	The Model that the shell set is in
title (read only)	string	The title for the shell set (or null if no title). This is only available if there is a ztf file for the model. If not null will be returned.
total (read only)	integer	The total number of shell items in the shell set
type (read only)	constant	The type for the shell set (will be Type.SET_SHELL)

Detailed Description

The SetShell class allows you to view sets in D3PLOT. There are various methods and properties available. This class requires a ztf file to be present for the model. See the documentation below for more details.

Details of functions

AllItems()

Description

Returns all of the shell items for the shell set in the model

Arguments

No arguments

Returns

array of Shell objects

Return type

Array

Example

To get the shell items in shell set s:

```
var items = s.AllItems();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on a shell set

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to clear on the shell set

Returns

No return value

Example

To clear flag f on shell set s:

```
s.ClearFlag();
```

First(model*[Model]*) [static]

Description

Returns the first shell set in the model (or null if there are no shell sets in the model)

Arguments

- **model** (Model)

Model to get first shell set in

Returns

SetShell object

Return type

SetShell

Example

To get the first shell set in model m:

```
var s = SetShell.First(m);
```

FlagAll(model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the shell sets in the model with a defined flag

Arguments

- **model** (Model)

Model that all the shell sets will be flagged in

- **flag** (Flag)

Flag (see AllocateFlag) to set on the shell sets

Returns

No return value

Example

To flag all of the shell sets with flag f in model m:

```
SetShell.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the shell set is flagged or not

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to test on the shell set

Returns

true if flagged, false if not

Return type

boolean

Example

To check if shell set s has flag f set on it:

```
if (s.Flagged(f) ) do_something...
```

GetAll(model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of SetShell objects or properties for all of the shell sets in the model. If the optional property argument is not given then an array of SetShell objects is returned. If the property argument is given, that property value for each shell set is returned in the array instead of a SetShell object

Arguments

- **model** (Model)

Model that all the shell sets are in

- **property (optional)** (string)

Name for property to get for all shell sets in the model

Returns

Array of SetShell objects or properties

Return type

Array

Example

To get all of the shell sets in model m:

```
var s = SetShell.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'label' for a shell set) for each shell set in model m:

```
var a = SetShell.GetAll(m, 'foo');
```

GetFlagged(model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Gets all of the shell sets in the model flagged with a defined flag. If the optional property argument is not given then an array of SetShell objects is returned. If the property argument is given, that property value for each shell set is returned in the array instead of a SetShell object

Arguments

- **model** (Model)

Model that the flagged shell sets are in

- **flag** (Flag)

Flag (see AllocateFlag) set on the shell sets to get

- **property (optional)** (string)

Name for property to get for all flagged shell sets in the model

Returns

Array of SetShell objects or properties

Return type

Array

Example

To get all of the shell sets flagged with flag f in model m:

```
SetShell.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'label' for a node) for all shell sets flagged with flag f in model m:

```
var a = SetShell.GetFlagged(m, f, 'foo');
```

GetFromID(model*[Model]*, label*[integer]*) [static]

Description

Returns the SetShell object for shell set in model with label (or null if it does not exist)

Arguments

- **model** (Model)

Model to get shell set in

- **label** (integer)

The Ansys LS-DYNA label for the shell set in the model

Returns

SetShell object

Return type

SetShell

Example

To get the shell set in model m with label 1000:

```
var s = SetShell.GetFromID(m, 1000);
```

GetFromIndex(model*[Model]*, index*[integer]*) [static]

Description

Returns the SetShell object for shell set in model with index (or null if it does not exist)

Arguments

- **model** (Model)

Model to get shell set in

- **index** (integer)

The D3PLOT internal index in the model for shell set, starting at 0.

Returns

SetShell object

Return type

SetShell

Example

To get the 51st shell set in model m:

```
var s = SetShell.GetFromIndex(m, 50);
```

Item(index*[integer]*)

Description

Returns a shell item from the shell set in the model

Arguments

- **index** (integer)

The index in the shell set to get the shell from (0 <= index < total)

Returns

Shell object

Return type

Shell

Example

To get the 10th shell in shell set s:

```
var items = s.Item(9);
```

Last(model*[Model]*) [static]

Description

Returns the last shell set in the model (or null if there are no shell sets in the model)

Arguments

- **model** (Model)

Model to get last shell set in

Returns

SetShell object

Return type

SetShell

Example

To get the last shell set in model m:

```
var s = SetShell.Last(m);
```

Next()

Description

Returns the next shell set in the model (or null if there is not one)

Arguments

No arguments

Returns

SetShell object

Return type

SetShell

Example

To get the next shell set after shell set s:

```
s = s.Next();
```

Previous()

Description

Returns the previous shell set in the model (or null if there is not one)

Arguments

No arguments

Returns

SetShell object

Return type

SetShell

Example

To get the previous shell set before shell set s:

```
s = s.Previous();
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on a shell set

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to set on the shell set

Returns

No return value

Example

To set flag f on shell set s:

```
s.SetFlag(f);
```

Total(model*[Model]*) [static]

Description

Returns the total number of shell sets in the model

Arguments

- **model** (Model)

Model to get total in

Returns

The number of shell sets

Return type

integer

Example

To get the number of shell sets in model m:

```
var total = SetShell.Total(m);
```

UnflagAll(model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the shell sets in the model

Arguments

- **model** (Model)

Model that the defined flag for all shell sets will be unset in

- **flag** (Flag)

Flag (see AllocateFlag) to unset on the shell sets

Returns

No return value

Example

To unset flag f on all of the shell sets in model m:

```
SetShell.UnflagAll(m, f);
```

SetSolid class

The SetSolid class gives you access to solid sets in D3PLOT. More...

The D3PLOT JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- First(model*[Model]*)
- FlagAll(model*[Model]*, flag*[Flag]*)
- GetAll(model*[Model]*, property (optional)*[string]*)
- GetFlagged(model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(model*[Model]*, label*[integer]*)
- GetFromIndex(model*[Model]*, index*[integer]*)
- Last(model*[Model]*)
- Total(model*[Model]*)
- UnflagAll(model*[Model]*, flag*[Flag]*)

Member functions

- AllItems()
- ClearFlag(flag*[Flag]*)
- Flagged(flag*[Flag]*)
- Item(index*[integer]*)
- Next()
- Previous()
- SetFlag(flag*[Flag]*)

SetSolid properties

Name	Type	Description
include (read only)	integer	The include file number in the model that the solid set is in
index (read only)	integer	The internal index for the solid set in D3PLOT (starting at 0)
label (read only)	integer	The Ansys LS-DYNA label for the solid set
model (read only)	Model	The Model that the solid set is in
title (read only)	string	The title for the solid set (or null if no title). This is only available if there is a ztf file for the model. If not null will be returned.
total (read only)	integer	The total number of solid items in the solid set
type (read only)	constant	The type for the solid set (will be Type.SET_SOLID)

Detailed Description

The SetSolid class allows you to view sets in D3PLOT. There are various methods and properties available. This class requires a ztf file to be present for the model. See the documentation below for more details.

Details of functions

AllItems()

Description

Returns all of the solid items for the solid set in the model

Arguments

No arguments

Returns

array of Solid objects

Return type

Array

Example

To get the solid items in solid set s:

```
var items = s.AllItems();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on a solid set

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to clear on the solid set

Returns

No return value

Example

To clear flag f on solid set s:

```
s.ClearFlag();
```

First(model*[Model]*) [static]

Description

Returns the first solid set in the model (or null if there are no solid sets in the model)

Arguments

- **model** (Model)

Model to get first solid set in

Returns

SetSolid object

Return type

SetSolid

Example

To get the first solid set in model m:

```
var s = SetSolid.First(m);
```

FlagAll(model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the solid sets in the model with a defined flag

Arguments

- **model** (Model)

Model that all the solid sets will be flagged in

- **flag** (Flag)

Flag (see AllocateFlag) to set on the solid sets

Returns

No return value

Example

To flag all of the solid sets with flag f in model m:

```
SetSolid.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the solid set is flagged or not

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to test on the solid set

Returns

true if flagged, false if not

Return type

boolean

Example

To check if solid set s has flag f set on it:

```
if (s.Flagged(f) ) do_something...
```

GetAll(model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of SetSolid objects or properties for all of the solid sets in the model. If the optional property argument is not given then an array of SetSolid objects is returned. If the property argument is given, that property value for each solid set is returned in the array instead of a SetSolid object

Arguments

- **model** (Model)

Model that all the solid sets are in

- **property (optional)** (string)

Name for property to get for all solid sets in the model

Returns

Array of SetSolid objects or properties

Return type

Array

Example

To get all of the solid sets in model m:

```
var s = SetSolid.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'label' for a solid set) for each solid set in model m:

```
var a = SetSolid.GetAll(m, 'foo');
```

GetFlagged(model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Gets all of the solid sets in the model flagged with a defined flag. If the optional property argument is not given then an array of SetSolid objects is returned. If the property argument is given, that property value for each solid set is returned in the array instead of a SetSolid object

Arguments

- **model** (Model)

Model that the flagged solid sets are in

- **flag** (Flag)

Flag (see AllocateFlag) set on the solid sets to get

- **property (optional)** (string)

Name for property to get for all flagged solid sets in the model

Returns

Array of SetSolid objects or properties

Return type

Array

Example

To get all of the solid sets flagged with flag f in model m:

```
SetSolid.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'label' for a node) for all solid sets flagged with flag f in model m:

```
var a = SetSolid.GetFlagged(m, f, 'foo');
```

GetFromID(model*[Model]*, label*[integer]*) [static]

Description

Returns the SetSolid object for solid set in model with label (or null if it does not exist)

Arguments

- **model** (Model)

Model to get solid set in

- **label** (integer)

The Ansys LS-DYNA label for the solid set in the model

Returns

SetSolid object

Return type

SetSolid

Example

To get the solid set in model m with label 1000:

```
var s = SetSolid.GetFromID(m, 1000);
```

GetFromIndex(model*[Model]*, index*[integer]*) [static]

Description

Returns the SetSolid object for solid set in model with index (or null if it does not exist)

Arguments

- **model** (Model)

Model to get solid set in

- **index** (integer)

The D3PLOT internal index in the model for solid set, starting at 0.

Returns

SetSolid object

Return type

SetSolid

Example

To get the 51st solid set in model m:

```
var s = SetSolid.GetFromIndex(m, 50);
```

Item(index*[integer]*)

Description

Returns a solid item from the solid set in the model

Arguments

- **index** (integer)

The index in the solid set to get the solid from (0 <= index < total)

Returns

Solid object

Return type

Solid

Example

To get the 10th solid in solid set s:

```
var items = s.Item(9);
```

Last(model*[Model]*) [static]

Description

Returns the last solid set in the model (or null if there are no solid sets in the model)

Arguments

- **model** (Model)

Model to get last solid set in

Returns

SetSolid object

Return type

SetSolid

Example

To get the last solid set in model m:

```
var s = SetSolid.Last(m);
```

Next()

Description

Returns the next solid set in the model (or null if there is not one)

Arguments

No arguments

Returns

SetSolid object

Return type

SetSolid

Example

To get the next solid set after solid set s:

```
s = s.Next();
```

Previous()

Description

Returns the previous solid set in the model (or null if there is not one)

Arguments

No arguments

Returns

SetSolid object

Return type

SetSolid

Example

To get the previous solid set before solid set s:

```
s = s.Previous();
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on a solid set

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to set on the solid set

Returns

No return value

Example

To set flag f on solid set s:

```
s.SetFlag(f);
```

Total(model*[Model]*) [static]

Description

Returns the total number of solid sets in the model

Arguments

- **model** (Model)

Model to get total in

Returns

The number of solid sets

Return type

integer

Example

To get the number of solid sets in model m:

```
var total = SetSolid.Total(m);
```

UnflagAll(model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the solid sets in the model

Arguments

- **model** (Model)

Model that the defined flag for all solid sets will be unset in

- **flag** (Flag)

Flag (see AllocateFlag) to unset on the solid sets

Returns

No return value

Example

To unset flag f on all of the solid sets in model m:

```
SetSolid.UnflagAll(m, f);
```

SetTshell class

The SetTshell class gives you access to thick shell sets in D3PLOT. More...

The D3PLOT JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- First(model*[Model]*)
- FlagAll(model*[Model]*, flag*[Flag]*)
- GetAll(model*[Model]*, property (optional)*[string]*)
- GetFlagged(model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(model*[Model]*, label*[integer]*)
- GetFromIndex(model*[Model]*, index*[integer]*)
- Last(model*[Model]*)
- Total(model*[Model]*)
- UnflagAll(model*[Model]*, flag*[Flag]*)

Member functions

- AllItems()
- ClearFlag(flag*[Flag]*)
- Flagged(flag*[Flag]*)
- Item(index*[integer]*)
- Next()
- Previous()
- SetFlag(flag*[Flag]*)

SetTshell properties

Name	Type	Description
include (read only)	integer	The include file number in the model that the thick shell set is in
index (read only)	integer	The internal index for the thick shell set in D3PLOT (starting at 0)
label (read only)	integer	The Ansys LS-DYNA label for the thick shell set
model (read only)	Model	The Model that the thick shell set is in
title (read only)	string	The title for the thick shell set (or null if no title). This is only available if there is a ztf file for the model. If not null will be returned.
total (read only)	integer	The total number of thick shell items in the thick shell set
type (read only)	constant	The type for the thick shell set (will be Type.SET_TSHELL)

Detailed Description

The SetTshell class allows you to view sets in D3PLOT. There are various methods and properties available. This class requires a ztf file to be present for the model. See the documentation below for more details.

Details of functions

AllItems()

Description

Returns all of the thick shell items for the thick shell set in the model

Arguments

No arguments

Returns

array of Tshell objects

Return type

Array

Example

To get the thick shell items in thick shell set s:

```
var items = s.AllItems();
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on a thick shell set

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to clear on the thick shell set

Returns

No return value

Example

To clear flag f on thick shell set s:

```
s.ClearFlag();
```

First(model*[Model]*) [static]

Description

Returns the first thick shell set in the model (or null if there are no thick shell sets in the model)

Arguments

- **model** (Model)

Model to get first thick shell set in

Returns

SetTshell object

Return type

SetTshell

Example

To get the first thick shell set in model m:

```
var s = SetTshell.First(m);
```

FlagAll(model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the thick shell sets in the model with a defined flag

Arguments

- **model** (Model)

Model that all the thick shell sets will be flagged in

- **flag** (Flag)

Flag (see AllocateFlag) to set on the thick shell sets

Returns

No return value

Example

To flag all of the thick shell sets with flag f in model m:

```
SetTshell.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the thick shell set is flagged or not

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to test on the thick shell set

Returns

true if flagged, false if not

Return type

boolean

Example

To check if thick shell set s has flag f set on it:

```
if (s.Flagged(f) ) do_something...
```

GetAll(model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of SetTshell objects or properties for all of the thick shell sets in the model. If the optional property argument is not given then an array of SetTshell objects is returned. If the property argument is given, that property value for each thick shell set is returned in the array instead of a SetTshell object

Arguments

- **model** (Model)

Model that all the thick shell sets are in

- **property (optional)** (string)

Name for property to get for all thick shell sets in the model

Returns

Array of SetTshell objects or properties

Return type

Array

Example

To get all of the thick shell sets in model m:

```
var s = SetTshell.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'label' for a thick shell set) for each thick shell set in model m:

```
var a = SetTshell.GetAll(m, 'foo');
```

GetFlagged(model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Gets all of the thick shell sets in the model flagged with a defined flag. If the optional property argument is not given then an array of SetTshell objects is returned. If the property argument is given, that property value for each thick shell set is returned in the array instead of a SetTshell object

Arguments

- **model** (Model)

Model that the flagged thick shell sets are in

- **flag** (Flag)

Flag (see AllocateFlag) set on the thick shell sets to get

- **property (optional)** (string)

Name for property to get for all flagged thick shell sets in the model

Returns

Array of SetTshell objects or properties

Return type

Array

Example

To get all of the thick shell sets flagged with flag f in model m:

```
SetTshell.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'label' for a node) for all thick shell sets flagged with flag f in model m:

```
var a = SetTshell.GetFlagged(m, f, 'foo');
```

GetFromID(model*[Model]*, label*[integer]*) [static]

Description

Returns the SetTshell object for thick shell set in model with label (or null if it does not exist)

Arguments

- **model** (Model)

Model to get thick shell set in

- **label** (integer)

The Ansys LS-DYNA label for the thick shell set in the model

Returns

SetTshell object

Return type

SetTshell

Example

To get the thick shell set in model m with label 1000:

```
var s = SetTshell.GetFromID(m, 1000);
```

GetFromIndex(model*[Model]*, index*[integer]*) [static]

Description

Returns the SetTshell object for thick shell set in model with index (or null if it does not exist)

Arguments

- **model** (Model)

Model to get thick shell set in

- **index** (integer)

The D3PLOT internal index in the model for thick shell set, starting at 0.

Returns

SetTshell object

Return type

SetTshell

Example

To get the 51st thick shell set in model m:

```
var s = SetTshell.GetFromIndex(m, 50);
```

Item(index*[integer]*)

Description

Returns a thick shell item from the thick shell set in the model

Arguments

- **index** (integer)

The index in the thick shell set to get the thick shell from (0 <= index < total)

Returns

Tshell object

Return type

Tshell

Example

To get the 10th thick shell in thick shell set s:

```
var items = s.Item(9);
```

Last(model*[Model]*) [static]

Description

Returns the last thick shell set in the model (or null if there are no thick shell sets in the model)

Arguments

- **model** (Model)

Model to get last thick shell set in

Returns

SetTshell object

Return type

SetTshell

Example

To get the last thick shell set in model m:

```
var s = SetTshell.Last(m);
```

Next()

Description

Returns the next thick shell set in the model (or null if there is not one)

Arguments

No arguments

Returns

SetTshell object

Return type

SetTshell

Example

To get the next thick shell set after thick shell set s:

```
s = s.Next();
```

Previous()

Description

Returns the previous thick shell set in the model (or null if there is not one)

Arguments

No arguments

Returns

SetTshell object

Return type

SetTshell

Example

To get the previous thick shell set before thick shell set s:

```
s = s.Previous();
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on a thick shell set

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to set on the thick shell set

Returns

No return value

Example

To set flag f on thick shell set s:

```
s.SetFlag(f);
```

Total(model*[Model]*) [static]

Description

Returns the total number of thick shell sets in the model

Arguments

- **model** (Model)

Model to get total in

Returns

The number of thick shell sets

Return type

integer

Example

To get the number of thick shell sets in model m:

```
var total = SetTshell.Total(m);
```

UnflagAll(model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the thick shell sets in the model

Arguments

- **model** (Model)

Model that the defined flag for all thick shell sets will be unset in

- **flag** (Flag)

Flag (see AllocateFlag) to unset on the thick shell sets

Returns

No return value

Example

To unset flag f on all of the thick shell sets in model m:

```
SetTshell.UnflagAll(m, f);
```

Shell class

The Shell class gives you access to shell elements in D3PLOT. More...

The D3PLOT JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(window*[GraphicsWindow]*, model*[Model]*)
- BlankFlagged(window*[GraphicsWindow]*, model*[Model]*, flag*[Flag]*)
- First(model*[Model]*)
- FlagAll(model*[Model]*, flag*[Flag]*)
- GetAll(model*[Model]*, property (optional)*[string]*)
- GetFlagged(model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(model*[Model]*, label*[integer]*)
- GetFromIndex(model*[Model]*, index*[integer]*)
- GetMultipleData(component*[constant]*, items*[array]*, options (optional)*[object]*)
- Last(model*[Model]*)
- Pick()
- Select(flag*[Flag]*)
- Total(model*[Model]*)
- TotalDeleted(model*[Model]*)
- UnblankAll(window*[GraphicsWindow]*, model*[Model]*)
- UnblankFlagged(window*[GraphicsWindow]*, model*[Model]*, flag*[Flag]*)
- UnflagAll(model*[Model]*, flag*[Flag]*)

Member functions

- Blank(window*[GraphicsWindow]*)
- Blanked(window*[GraphicsWindow]*)
- ClearFlag(flag*[Flag]*)
- Deleted()
- Flagged(flag*[Flag]*)
- GetData(component*[constant]*, options (optional)*[object]*)
- LocalAxes()
- Next()
- PlasticStrain(options (optional)*[object]*)
- Previous()
- SetFlag(flag*[Flag]*)
- StrainTensor(options (optional)*[object]*)
- StressTensor(options (optional)*[object]*)
- Topology()
- Unblank(window*[GraphicsWindow]*)
- VonMisesStress(options (optional)*[object]*)

Shell properties

Name	Type	Description
colour	Colour	The colour of the shell
data (read only)	real|array	Component data for a shell passed as an argument to GetMultipleData. Note that data will only exist for the instance of the shell passed to GetMultipleData. i.e. it is a local property stored on the specific instance. It is not stored in the D3PLOT database
displayMode	integer	The display mode of the shell. Can be Constant.WIREFRAME, Constant.HIDDEN, Constant.SHADED or Constant.CURRENT.

include (read only)	integer	The include file number in the model that the shell is in
index (read only)	integer	The internal index for the shell in D3PLOT (starting at 0)
integrationPoints (read only)	integer	The number of through thickness integration points that the shell has
label (read only)	integer	The Ansys LS-DYNA label for the shell
material (read only)	Material	The Material the shell has. This is only available if there is a ztf file for the model. If not null will be returned. If this is a PART_COMPOSITE then null will be returned. If the model comes from writing a PTF/d3plot file from PRIMER and the material was latent there, null will be returned. Part.GetCompositeData should be used to get material data in this case
model (read only)	Model	The Model that the shell is in
onPlanIntegrationPoints (read only)	integer	The number of on plan integration points that the shell has
overlayColour	Colour	The overlay colour of the shell
overlayMode	integer	The overlay mode of the shell. Can be Constant.NO_OVERLAY, Constant.FREE_EDGE, Constant.ALL_EDGES or Constant.CURRENT.
part (read only)	Part	The Part the shell is in
transparency	integer	The transparency of the shell between 0 (opaque) and 100 (clear)
type (read only)	constant	The type for the shell (will be Type.SHELL)

Detailed Description

The Shell class allows you to inspect shell elements in a model. See the documentation below for more details.

Details of functions

Blank(window*[GraphicsWindow]*)

Description

Blanks the shell in a graphics window

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to blank the shell in

Returns

No return value

Example

To blank shell s in graphics window g:

```
s.Blank(g);
```

BlankAll(window*[GraphicsWindow]*, model*[Model]*) [static]

Description

Blanks all of the shells in the model

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to blank the shells in

- **model** (Model)

Model that all the shells will be blanked in

Returns

No return value

Example

To blank all of the shells in model m, in graphics window gw:

```
Shell.BlankAll(gw, m);
```

BlankFlagged(window*[GraphicsWindow]*, model*[Model]*, flag*[Flag]*) [static]

Description

Blanks all of the shells in the model flagged with a defined flag

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to blank the shells in

- **model** (Model)

Model that the flagged shells will be blanked in

- **flag** (Flag)

Flag (see AllocateFlag) set on the shells to blank

Returns

No return value

Example

To blank all of the shells flagged with flag f in model m, in graphics window gw:

```
Shell.BlankFlagged(gw, m, f);
```

Blanked(window*[GraphicsWindow]*)

Description

Checks if the shell is blanked in a graphics window or not

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) in which to check if the shell is blanked

Returns

true if blanked, false if not

Return type

boolean

Example

To check if shell s is blanked in graphics window g:

```
if (s.Blanked(g) ) do_something...
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on a shell

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to clear on the shell

Returns

No return value

Example

To clear flag f on shell s:

```
s.ClearFlag();
```

Deleted()

Description

Checks if the shell has been deleted or not

Arguments

No arguments

Returns

true if deleted, false if not

Return type

boolean

Example

To check if shell s has been deleted:

```
if (s.Deleted() ) do_something...
```

First(model*[Model]*) [static]

Description

Returns the first shell in the model (or null if there are no shells in the model)

Arguments

- **model** (Model)

Model to get first shell in

Returns

Shell object

Return type

Shell

Example

To get the first shell in model m:

```
var s = Shell.First(m);
```

FlagAll(model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the shells in the model with a defined flag

Arguments

- **model** (Model)

Model that all the shells will be flagged in

- **flag** (Flag)

Flag (see AllocateFlag) to set on the shells

Returns

No return value

Example

To flag all of the shells with flag f in model m:

```
Shell.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the shell is flagged or not

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to test on the shell

Returns

true if flagged, false if not

Return type

boolean

Example

To check if shell s has flag f set on it:

```
if (s.Flagged(f) ) do_something...
```

GetAll(model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Shell objects or properties for all of the shells in the model. If the optional property argument is not given then an array of Shell objects is returned. If the property argument is given, that property value for each shell is returned in the array instead of a Shell object

Arguments

- **model** (Model)

Model that all the shells are in

- **property (optional)** (string)

Name for property to get for all shells in the model

Returns

Array of Shell objects or properties

Return type

Array

Example

To get all of the shells in model m:

```
var s = Shell.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'label' for a shell) for each shell in model m:

```
var a = Shell.GetAll(m, 'foo');
```

GetData(component*[constant]*, options (optional)*[object]*)

Description

Returns the value for a data component.
Also see GetMultipleData

Arguments

- **component** (constant)

Component constant to get data for

- **options (optional)** (object)

Object containing options for getting data. Can be any of:

Object has the following properties:

Name	Type	Description
extra	integer	The extra data component number if component Component.SOX for solids, Component.BMX for beams or Component.SHX for shells and thick shells
ip	integer	Integration point number to get the data at (ip >= 1 or one of the constants Constant.TOP, Constant.MIDDLE or Constant.BOTTOM). If the integration point is not defined it will use the integration point defined on the current GUI "data" panel, which defaults to the middle surface for shells, thick shells, and solids, and Mag All for beams, but may vary if changed by an interactive user. If consistent output from a script is required, independent of any prior interactive activity, an explicit integration point or surface should be defined
op	integer	On plane integration point number for shells and thick shells (op >= 1 [default])

referenceFrame	constant	The frame of reference to return values in. Either Constant.GLOBAL (default), Constant.LOCAL, Constant.CYLINDRICAL, Constant.USER_DEFINED or Constant.MATERIAL. This is only necessary for directional components (eg X stress) and then only when something other than the default Constant.GLOBAL coordinate system is to be used
user	integer	The user-defined component number if component Component.UNOS, Component.UNOV, Component.USSS, Component.USST, Component.UBMS or Component.UBMV

Returns

Number if a scalar component, array if a vector or tensor component (or null if the value cannot be calculated because it's not available in the model).
If requesting an invalid component it will throw an error (e.g. Component.AREA of a node).

Return type

real|array

Example

To calculate a component and check it has been calculated (note that in the example, the argument extra is optional):

```
var value = s.GetData(component, {extra: 1});
if (value !== null) do_something...
```

GetFlagged(model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Gets all of the shells in the model flagged with a defined flag. If the optional property argument is not given then an array of Shell objects is returned. If the property argument is given, that property value for each shell is returned in the array instead of a Shell object

Arguments

- **model** (Model)

Model that the flagged shells are in

- **flag** (Flag)

Flag (see AllocateFlag) set on the shells to get

- **property (optional)** (string)

Name for property to get for all flagged shells in the model

Returns

Array of Shell objects or properties

Return type

Array

Example

To get all of the shells flagged with flag f in model m:

```
Shell.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'label' for a node) for all shells flagged with flag f in model m:

```
var a = Shell.GetFlagged(m, f, 'foo');
```

GetFromID(model*[Model]*, label*[integer]*) [static]

Description

Returns the Shell object for shell in model with label (or null if it does not exist)

Arguments

- **model** (Model)

Model to get shell in

- **label** (integer)

The Ansys LS-DYNA label for the shell in the model

Returns

Shell object

Return type

Shell

Example

To get the shell in model m with label 1000:

```
var s = Shell.GetFromID(m, 1000);
```

GetFromIndex(model*[Model]*, index*[integer]*) [static]

Description

Returns the Shell object for shell in model with index (or null if it does not exist)

Arguments

- **model** (Model)

Model to get shell in

- **index** (integer)

The D3PLOT internal index in the model for shell, starting at 0.

Returns

Shell object

Return type

Shell

Example

To get the 51st shell in model m:

```
var s = Shell.GetFromIndex(m, 50);
```

GetMultipleData(component*[constant]*, items*[array]*, options (optional)*[object]*) [static]

Description

Returns the value for a data component for multiple shells. For each shell a local property called data will be created containing a number if a scalar component, or an array if a vector or tensor component (or null if the value cannot be calculated). The data is also returned as an object.
Also see GetData

Arguments

- **component** (constant)

Component constant to get data for

- **items** (array)

Array of Shell objects to get the data for. All of the shells must be in the same model.

- **options (optional)** (object)

Object containing options for getting data. Can be any of:

Object has the following properties:

Name	Type	Description
extra	integer	The extra data component number if component Component.SOX for solids, Component.BMX for beams or Component.SHX for shells and thick shells
ip	integer	Integration point number to get the data at (ip >= 1 or one of the constants Constant.TOP, Constant.MIDDLE or Constant.BOTTOM)
op	integer	On plane integration point number for shells and thick shells (op >= 1 [default])
referenceFrame	constant	The frame of reference to return values in. Either Constant.GLOBAL (default), Constant.LOCAL, Constant.CYLINDRICAL, Constant.USER_DEFINED or Constant.MATERIAL. This is only necessary for directional components (eg X stress) and then only when something other than the default Constant.GLOBAL coordinate system is to be used
user	integer	The user-defined component number if component Component.UNOS, Component.UNOV, Component.USSS, Component.USST, Component.UBMS or Component.UBMV

Returns

Object containing the data. A property is created in the object for each shell with the label. The value of the property is a number if a scalar component or an array if a vector or tensor component (or null if the value cannot be calculated)

Return type

object

Example

To calculate a component for shells in array items and use the data property (note that in the example, the argument extra is optional):

```
Shell.GetMultipleData(component, items, {extra: 1});
for (i=0; i<items.length; i++)
{
    if (items[i].data !== null) do_something...
}
```

To calculate a component for shells in array items and use the return value (note that in the example, the argument extra is optional):

```
var data = Shell.GetMultipleData(component, items, {extra: 1});
for (d in data)
{
    Message("Label is " + d);
    if (data[d] !== null) do_something...
}
```

Last(model[*Model*]) [static]

Description

Returns the last shell in the model (or null if there are no shells in the model)

Arguments

- **model** (Model)

Model to get last shell in

Returns

Shell object

Return type

Shell

Example

To get the last shell in model m:

```
var s = Shell.Last(m);
```

LocalAxes()

Description

Returns the local axes of the element in model space, expressed as direction cosines in a 2D array. Beam elements must have 3 nodes to be able to return local axes.

Arguments

No arguments

Returns

array of arrays

Return type

Array

Example

To get the local axes for shell s:

```
var axes = s.LocalAxes();
var xAxis = [ axes[0][0], axes[0][1], axes[0][2] ];
var yAxis = [ axes[1][0], axes[1][1], axes[1][2] ];
var zAxis = [ axes[2][0], axes[2][1], axes[2][2] ];
```

Next()

Description

Returns the next shell in the model (or null if there is not one)

Arguments

No arguments

Returns

Shell object

Return type

Shell

Example

To get the next shell after shell s:

```
s = s.Next();
```

Pick() [static]

Description

Allows the user to pick a shell from the screen

Arguments

No arguments

Returns

Shell object or null if cancelled

Return type

Shell

Example

To pick a shell:

```
var s = Shell.Pick();
```

PlasticStrain(options (optional)*[object]*)

Description

Returns the effective plastic strain for the shell

Arguments

- **options (optional)** (object)

Object containing options for getting data. Can be any of:

Object has the following properties:

Name	Type	Description
ip	integer	Integration point number to get the data at (ip >= 1 or one of the constants Constant.TOP, Constant.MIDDLE or Constant.BOTTOM)
op	integer	On plane integration point number for shells and thick shells (op >= 1 [default])

Returns

Plastic strain (or null if the value cannot be calculated)

Return type

real

Example

To return the effective plastic strain of shell s:

```
var strain = s.PlasticStrain();
if (strain !== null) do_something...
```

Previous()

Description

Returns the previous shell in the model (or null if there is not one)

Arguments

No arguments

Returns

Shell object

Return type

Shell

Example

To get the previous shell before shell s:

```
s = s.Previous();
```

Select(flag*[Flag]*) [static]

Description

Selects shells using an object menu

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to use when selecting shells

Returns

The number of shells selected or null if menu cancelled

Return type

integer

Example

To select shells, flagging those selected with flag f:

```
var total = Shell.Select(f);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on a shell

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to set on the shell

Returns

No return value

Example

To set flag f on shell s:

```
s.SetFlag(f);
```

StrainTensor(options (optional)*[object]*)

Description

Returns the strain tensor for the shell

Arguments

- **options (optional)** (object)

Object containing options for getting data. Can be any of:

Object has the following properties:

Name	Type	Description
ip	integer	Integration point number to get the data at (ip >= 1 or one of the constants Constant.TOP, Constant.MIDDLE or Constant.BOTTOM)
op	integer	On plane integration point number for shells and thick shells (op >= 1 [default])
referenceFrame	constant	The frame of reference to return values in. Either Constant.GLOBAL (default), Constant.LOCAL, Constant.CYLINDRICAL, Constant.USER_DEFINED or Constant.MATERIAL

Returns

Array containing the strain tensor [Exx, Eyy, Ezz, Exy, Eyz, Ezx] (or null if the value cannot be calculated)

Return type

array

Example

To return the strain tensor of shell s:

```
var tensor = s.StrainTensor();
if (tensor !== null) do_something...
```

StressTensor(options (optional)*[object]*)

Description

Returns the stress tensor for the shell

Arguments

- **options (optional)** (object)

Object containing options for getting data. Can be any of:

Object has the following properties:

Name	Type	Description
ip	integer	Integration point number to get the data at (ip >= 1 or one of the constants Constant.TOP, Constant.MIDDLE or Constant.BOTTOM)
op	integer	On plane integration point number for shells and thick shells (op >= 1 [default])

referenceFrame	constant	The frame of reference to return values in. Either Constant.GLOBAL (default), Constant.LOCAL, Constant.CYLINDRICAL, Constant.USER_DEFINED or Constant.MATERIAL

Returns

Array containing the stress tensor [Exx, Eyy, Ezz, Exy, Eyz, Ezx] (or null if the value cannot be calculated)

Return type

array

Example

To return the stress tensor of shell s:

```
var tensor = s.StressTensor();
if (tensor !== null) do_something...
```

Topology()

Description

Returns the topology for the shell in the model

Arguments

No arguments

Returns

array of Node objects

Return type

Array

Example

To get the topology for shell s:

```
var topology = s.Topology();
```

Total(model[*Model*]) [static]

Description

Returns the total number of shells in the model

Arguments

- **model** (Model)

Model to get total in

Returns

The number of shells

Return type

integer

Example

To get the number of shells in model m:

```
var total = Shell.Total(m);
```

TotalDeleted(model*[Model]*) [static]

Description

Returns the total number of shells that have been deleted in a model in the state given by its state property

Arguments

- **model** (Model)

Model to get total in

Returns

The number of shells that have been deleted

Return type

integer

Example

To get the number of shells in model m that have been deleted in state 10:

```
m.state = 10;
var total = Shell.TotalDeleted(m);
```

Unblank(window*[GraphicsWindow]*)

Description

Unblanks the shell in a graphics window

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to unblank the shell in

Returns

No return value

Example

To unblank shell s in graphics window g:

```
s.Unblank(g);
```

UnblankAll(window*[GraphicsWindow]*, model*[Model]*) [static]

Description

Unblanks all of the shells in the model

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to unblank the shells in

- **model** (Model)

Model that all the shells will be unblanked in

Returns

No return value

Example

To unblank all of the shells in model m, in graphics window gw:

```
Shell.UnblankAll(gw, m);
```

UnblankFlagged(window*[GraphicsWindow]*, model*[Model]*, flag*[Flag]*) [static]

Description

Unblanks all of the shells in the model flagged with a defined flag

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to unblank the shells in

- **model** (Model)

Model that the flagged shells will be unblanked in

- **flag** (Flag)

Flag (see AllocateFlag) set on the shells to unblank

Returns

No return value

Example

To unblank all of the shells flagged with flag f in model m, in graphics window gw:

```
Shell.UnblankFlagged(gw, m, f);
```

UnflagAll(model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the shells in the model

Arguments

- **model** (Model)

Model that the defined flag for all shells will be unset in

- **flag** (Flag)

Flag (see AllocateFlag) to unset on the shells

Returns

No return value

Example

To unset flag f on all of the shells in model m:

```
Shell.UnflagAll(m, f);
```

VonMisesStress(options (optional)*[object]*)

Description

Returns the von Mises stress for the shell (or null if the value cannot be calculated)

Arguments

- **options (optional)** (object)

Object containing options for getting data. Can be any of:

Object has the following properties:

Name	Type	Description
ip	integer	Integration point number to get the data at (ip >= 1 or one of the constants Constant.TOP, Constant.MIDDLE or Constant.BOTTOM)
op	integer	On plane integration point number for shells and thick shells (op >= 1 [default])

Returns

von Mises stress

Return type

real

Example

To return the von Mises stress of shell s:

```
var svm = s.VonMisesStress();
if (svm !== null) do_something...
```

Solid class

The Solid class gives you access to solid elements in D3PLOT. More...

The D3PLOT JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(window*[GraphicsWindow]*, model*[Model]*)
- BlankFlagged(window*[GraphicsWindow]*, model*[Model]*, flag*[Flag]*)
- First(model*[Model]*)
- FlagAll(model*[Model]*, flag*[Flag]*)
- GetAll(model*[Model]*, property (optional)*[string]*)
- GetFlagged(model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(model*[Model]*, label*[integer]*)
- GetFromIndex(model*[Model]*, index*[integer]*)
- GetMultipleData(component*[constant]*, items*[array]*, options (optional)*[object]*)
- Last(model*[Model]*)
- Pick()
- Select(flag*[Flag]*)
- Total(model*[Model]*)
- TotalDeleted(model*[Model]*)
- UnblankAll(window*[GraphicsWindow]*, model*[Model]*)
- UnblankFlagged(window*[GraphicsWindow]*, model*[Model]*, flag*[Flag]*)
- UnflagAll(model*[Model]*, flag*[Flag]*)

Member functions

- Blank(window*[GraphicsWindow]*)
- Blanked(window*[GraphicsWindow]*)
- ClearFlag(flag*[Flag]*)
- Deleted()
- Flagged(flag*[Flag]*)
- GetData(component*[constant]*, options (optional)*[object]*)
- LocalAxes()
- Next()
- PlasticStrain(options (optional)*[object]*)
- Previous()
- SetFlag(flag*[Flag]*)
- StrainTensor(options (optional)*[object]*)
- StressTensor(options (optional)*[object]*)
- Topology()
- Unblank(window*[GraphicsWindow]*)
- VonMisesStress(options (optional)*[object]*)

Solid properties

Name	Type	Description
colour	Colour	The colour of the solid
data (read only)	real|array	Component data for a solid passed as an argument to GetMultipleData. Note that data will only exist for the instance of the solid passed to GetMultipleData. i.e. it is a local property stored on the specific instance. It is not stored in the D3PLOT database
displayMode	integer	The display mode of the solid. Can be Constant.WIREFRAME, Constant.HIDDEN, Constant.SHADED or Constant.CURRENT.

include (read only)	integer	The include file number in the model that the solid is in
index (read only)	integer	The internal index for the solid in D3PLOT (starting at 0)
label (read only)	integer	The Ansys LS-DYNA label for the solid
material (read only)	Material	The Material the solid has. This is only available if there is a ztf file for the model. If not null will be returned. If this is a PART_COMPOSITE then null will be returned. If the model comes from writing a PTF/d3plot file from PRIMER and the material was latent there, null will be returned. Part.GetCompositeData should be used to get material data in this case
model (read only)	Model	The Model that the solid is in
overlayColour	Colour	The overlay colour of the solid
overlayMode	integer	The overlay mode of the solid. Can be Constant.NO_OVERLAY, Constant.FREE_EDGE, Constant.ALL_EDGES or Constant.CURRENT.
part (read only)	Part	The Part the solid is in
transparency	integer	The transparency of the solid between 0 (opaque) and 100 (clear)
type (read only)	constant	The type for the solid (will be Type.SOLID)

Detailed Description

The Solid class allows you to inspect solid elements in a model. See the documentation below for more details.

Details of functions

Blank(window*[GraphicsWindow]*)

Description

Blanks the solid in a graphics window

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to blank the solid in

Returns

No return value

Example

To blank solid s in graphics window g:

```
s.Blank(g);
```

BlankAll(window*[GraphicsWindow]*, model*[Model]*) [static]

Description

Blanks all of the solids in the model

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to blank the solids in

- **model** (Model)

Model that all the solids will be blanked in

Returns

No return value

Example

To blank all of the solids in model m, in graphics window gw:

```
Solid.BlankAll(gw, m);
```

BlankFlagged(window*[GraphicsWindow]*, model*[Model]*, flag*[Flag]*) [static]

Description

Blanks all of the solids in the model flagged with a defined flag

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to blank the solids in

- **model** (Model)

Model that the flagged solids will be blanked in

- **flag** (Flag)

Flag (see AllocateFlag) set on the solids to blank

Returns

No return value

Example

To blank all of the solids flagged with flag f in model m, in graphics window gw:

```
Solid.BlankFlagged(gw, m, f);
```

Blanked(window*[GraphicsWindow]*)

Description

Checks if the solid is blanked in a graphics window or not

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) in which to check if the solid is blanked

Returns

true if blanked, false if not

Return type

boolean

Example

To check if solid s is blanked in graphics window g:

```
if (s.Blanked(g) ) do_something...
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on a solid

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to clear on the solid

Returns

No return value

Example

To clear flag f on solid s:

```
s.ClearFlag();
```

Deleted()

Description

Checks if the solid has been deleted or not

Arguments

No arguments

Returns

true if deleted, false if not

Return type

boolean

Example

To check if solid s has been deleted:

```
if (s.Deleted() ) do_something...
```

First(model*[Model]*) [static]

Description

Returns the first solid in the model (or null if there are no solids in the model)

Arguments

- **model** (Model)

Model to get first solid in

Returns

Solid object

Return type

Solid

Example

To get the first solid in model m:

```
var s = Solid.First(m);
```

FlagAll(model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the solids in the model with a defined flag

Arguments

- **model** (Model)

Model that all the solids will be flagged in

- **flag** (Flag)

Flag (see AllocateFlag) to set on the solids

Returns

No return value

Example

To flag all of the solids with flag f in model m:

```
Solid.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the solid is flagged or not

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to test on the solid

Returns

true if flagged, false if not

Return type

boolean

Example

To check if solid s has flag f set on it:

```
if (s.Flagged(f) ) do_something...
```

GetAll(model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Solid objects or properties for all of the solids in the model. If the optional property argument is not given then an array of Solid objects is returned. If the property argument is given, that property value for each solid is returned in the array instead of a Solid object

Arguments

- **model** (Model)

Model that all the solids are in

- **property (optional)** (string)

Name for property to get for all solids in the model

Returns

Array of Solid objects or properties

Return type

Array

Example

To get all of the solids in model m:

```
var s = Solid.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'label' for a solid) for each solid in model m:

```
var a = Solid.GetAll(m, 'foo');
```

GetData(component*[constant]*, options (optional)*[object]*)

Description

Returns the value for a data component.
Also see GetMultipleData

Arguments

- **component** (constant)

Component constant to get data for

- **options (optional)** (object)

Object containing options for getting data. Can be any of:

Object has the following properties:

Name	Type	Description
extra	integer	The extra data component number if component Component.SOX for solids, Component.BMX for beams or Component.SHX for shells and thick shells
ip	integer	Integration point number to get the data at (ip >= 1 or one of the constants Constant.TOP, Constant.MIDDLE or Constant.BOTTOM). If the integration point is not defined it will use the integration point defined on the current GUI "data" panel, which defaults to the middle surface for shells, thick shells, and solids, and Mag All for beams, but may vary if changed by an interactive user. If consistent output from a script is required, independent of any prior interactive activity, an explicit integration point or surface should be defined
op	integer	On plane integration point number for shells and thick shells (op >= 1 [default])

referenceFrame	constant	The frame of reference to return values in. Either Constant.GLOBAL (default), Constant.LOCAL, Constant.CYLINDRICAL, Constant.USER_DEFINED or Constant.MATERIAL. This is only necessary for directional components (eg X stress) and then only when something other than the default Constant.GLOBAL coordinate system is to be used
user	integer	The user-defined component number if component Component.UNOS, Component.UNOV, Component.USSS, Component.USST, Component.UBMS or Component.UBMV

Returns

Number if a scalar component, array if a vector or tensor component (or null if the value cannot be calculated because it's not available in the model).
If requesting an invalid component it will throw an error (e.g. Component.AREA of a node).

Return type

real|array

Example

To calculate a component and check it has been calculated (note that in the example, the argument extra is optional):

```
var value = s.GetData(component, {extra: 1});
if (value !== null) do_something...
```

GetFlagged(model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Gets all of the solids in the model flagged with a defined flag. If the optional property argument is not given then an array of Solid objects is returned. If the property argument is given, that property value for each solid is returned in the array instead of a Solid object

Arguments

- **model** (Model)

Model that the flagged solids are in

- **flag** (Flag)

Flag (see AllocateFlag) set on the solids to get

- **property (optional)** (string)

Name for property to get for all flagged solids in the model

Returns

Array of Solid objects or properties

Return type

Array

Example

To get all of the solids flagged with flag f in model m:

```
Solid.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'label' for a node) for all solids flagged with flag f in model m:

```
var a = Solid.GetFlagged(m, f, 'foo');
```

GetFromID(model*[Model]*, label*[integer]*) [static]

Description

Returns the Solid object for solid in model with label (or null if it does not exist)

Arguments

- **model** (Model)

Model to get solid in

- **label** (integer)

The Ansys LS-DYNA label for the solid in the model

Returns

Solid object

Return type

Solid

Example

To get the solid in model m with label 1000:

```
var s = Solid.GetFromID(m, 1000);
```

GetFromIndex(model*[Model]*, index*[integer]*) [static]

Description

Returns the Solid object for solid in model with index (or null if it does not exist)

Arguments

- **model** (Model)

Model to get solid in

- **index** (integer)

The D3PLOT internal index in the model for solid, starting at 0.

Returns

Solid object

Return type

Solid

Example

To get the 51st solid in model m:

```
var s = Solid.GetFromIndex(m, 50);
```

GetMultipleData(component*[constant]*, items*[array]*, options (optional)*[object]*) [static]

Description

Returns the value for a data component for multiple solids. For each solid a local property called data will be created containing a number if a scalar component, or an array if a vector or tensor component (or null if the value cannot be calculated). The data is also returned as an object.
Also see GetData

Arguments

- **component** (constant)

Component constant to get data for

- **items** (array)

Array of Solid objects to get the data for. All of the solids must be in the same model.

- **options (optional)** (object)

Object containing options for getting data. Can be any of:

Object has the following properties:

Name	Type	Description
extra	integer	The extra data component number if component Component.SOX for solids, Component.BMX for beams or Component.SHX for shells and thick shells
ip	integer	Integration point number to get the data at (ip >= 1 or one of the constants Constant.TOP, Constant.MIDDLE or Constant.BOTTOM)
op	integer	On plane integration point number for shells and thick shells (op >= 1 [default])
referenceFrame	constant	The frame of reference to return values in. Either Constant.GLOBAL (default), Constant.LOCAL, Constant.CYLINDRICAL, Constant.USER_DEFINED or Constant.MATERIAL. This is only necessary for directional components (eg X stress) and then only when something other than the default Constant.GLOBAL coordinate system is to be used
user	integer	The user-defined component number if component Component.UNOS, Component.UNOV, Component.USSS, Component.USST, Component.UBMS or Component.UBMV

Returns

Object containing the data. A property is created in the object for each solid with the label. The value of the property is a number if a scalar component or an array if a vector or tensor component (or null if the value cannot be calculated)

Return type

object

Example

To calculate a component for solids in array items and use the data property (note that in the example, the argument extra is optional):

```
Solid.GetMultipleData(component, items, {extra: 1});
for (i=0; i<items.length; i++)
{
    if (items[i].data !== null) do_something...
}
```

To calculate a component for solids in array items and use the return value (note that in the example, the argument extra is optional):

```
var data = Solid.GetMultipleData(component, items, {extra: 1});
for (d in data)
{
    Message("Label is " + d);
    if (data[d] !== null) do_something...
}
```

Last(model[*Model*]) [static]

Description

Returns the last solid in the model (or null if there are no solids in the model)

Arguments

- **model** (Model)

Model to get last solid in

Returns

Solid object

Return type

Solid

Example

To get the last solid in model m:

```
var s = Solid.Last(m);
```

LocalAxes()

Description

Returns the local axes of the element in model space, expressed as direction cosines in a 2D array. Beam elements must have 3 nodes to be able to return local axes.

Arguments

No arguments

Returns

array of arrays

Return type

Array

Example

To get the local axes for solid s:

```
var axes = s.LocalAxes();
var xAxis = [ axes[0][0], axes[0][1], axes[0][2] ];
var yAxis = [ axes[1][0], axes[1][1], axes[1][2] ];
var zAxis = [ axes[2][0], axes[2][1], axes[2][2] ];
```

Next()

Description

Returns the next solid in the model (or null if there is not one)

Arguments

No arguments

Returns

Solid object

Return type

Solid

Example

To get the next solid after solid s:

```
s = s.Next();
```

Pick() [static]

Description

Allows the user to pick a solid from the screen

Arguments

No arguments

Returns

Solid object or null if cancelled

Return type

Solid

Example

To pick a solid:

```
var s = Solid.Pick();
```

PlasticStrain(options (optional)*[object]*)

Description

Returns the effective plastic strain for the solid (or null if the value cannot be calculated)

Arguments

- **options (optional)** (object)

Object containing options for getting data. Can be any of:

Object has the following properties:

Name	Type	Description
ip	integer	Integration point number to get the data at (ip >= 1)

Returns

Plastic strain

Return type

real

Example

To return the effective plastic strain of solid s:

```
var strain = s.PlasticStrain();
if (strain !== null) do_something...
```

Previous()

Description

Returns the previous solid in the model (or null if there is not one)

Arguments

No arguments

Returns

Solid object

Return type

Solid

Example

To get the previous solid before solid s:

```
s = s.Previous();
```

Select(flag*[Flag]*) [static]

Description

Selects solids using an object menu

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to use when selecting solids

Returns

The number of solids selected or null if menu cancelled

Return type

integer

Example

To select solids, flagging those selected with flag f:

```
var total = Solid.Select(f);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on a solid

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to set on the solid

Returns

No return value

Example

To set flag f on solid s:

```
s.SetFlag(f);
```

StrainTensor(options (optional)*[object]*)

Description

Returns the strain tensor for the solid

Arguments

- **options (optional)** (object)

Object containing options for getting data. Can be any of:

Object has the following properties:

Name	Type	Description
ip	integer	Integration point number to get the data at (ip >= 1)
referenceFrame	constant	The frame of reference to return values in. Either Constant.GLOBAL (default), Constant.LOCAL, Constant.CYLINDRICAL, Constant.USER_DEFINED or Constant.MATERIAL

Returns

Array containing the strain tensor [Exx, Eyy, Ezz, Exy, Eyz, Ezx] (or null if the value cannot be calculated)

Return type

array

Example

To return the strain tensor of solid s:

```
var tensor = s.StrainTensor();
if (tensor !== null) do_something...
```

StressTensor(options (optional)*[object]*)

Description

Returns the stress tensor for the solid

Arguments

- **options (optional)** (object)

Object containing options for getting data. Can be any of:

Object has the following properties:

Name	Type	Description
ip	integer	Integration point number to get the data at (ip >= 1)
referenceFrame	constant	The frame of reference to return values in. Either Constant.GLOBAL (default), Constant.LOCAL, Constant.CYLINDRICAL, Constant.USER_DEFINED or Constant.MATERIAL

Returns

Array containing the stress tensor [Exx, Eyy, Ezz, Exy, Eyz, Ezx] (or null if the value cannot be calculated)

Return type

array

Example

To return the stress tensor of solid s:

```
var tensor = s.StressTensor();
if (tensor !== null) do_something...
```

Topology()

Description

Returns the topology for the solid in the model

Arguments

No arguments

Returns

array of Node objects

Return type

Array

Example

To get the topology for solid s:

```
var topology = s.Topology();
```

Total(model[*Model*]) [static]

Description

Returns the total number of solids in the model

Arguments

- **model** (Model)

Model to get total in

Returns

The number of solids

Return type

integer

Example

To get the number of solids in model m:

```
var total = Solid.Total(m);
```

TotalDeleted(model*[Model]*) [static]

Description

Returns the total number of solids that have been deleted in a model in the state given by its state property

Arguments

- **model** (Model)

Model to get total in

Returns

The number of solids that have been deleted

Return type

integer

Example

To get the number of solids in model m that have been deleted in state 10:

```
m.state = 10;
var total = Solid.TotalDeleted(m);
```

Unblank(window*[GraphicsWindow]*)

Description

Unblanks the solid in a graphics window

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to unblank the solid in

Returns

No return value

Example

To unblank solid s in graphics window g:

```
s.Unblank(g);
```

UnblankAll(window*[GraphicsWindow]*, model*[Model]*) [static]

Description

Unblanks all of the solids in the model

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to unblank the solids in

- **model** (Model)

Model that all the solids will be unblanked in

Returns

No return value

Example

To unblank all of the solids in model m, in graphics window gw:

```
Solid.UnblankAll(gw, m);
```

UnblankFlagged(window*[GraphicsWindow]*, model*[Model]*, flag*[Flag]*) [static]

Description

Unblanks all of the solids in the model flagged with a defined flag

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to unblank the solids in

- **model** (Model)

Model that the flagged solids will be unblanked in

- **flag** (Flag)

Flag (see AllocateFlag) set on the solids to unblank

Returns

No return value

Example

To unblank all of the solids flagged with flag f in model m, in graphics window gw:

```
Solid.UnblankFlagged(gw, m, f);
```

UnflagAll(model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the solids in the model

Arguments

- **model** (Model)

Model that the defined flag for all solids will be unset in

- **flag** (Flag)

Flag (see AllocateFlag) to unset on the solids

Returns

No return value

Example

To unset flag f on all of the solids in model m:

```
Solid.UnflagAll(m, f);
```

VonMisesStress(options (optional)*[object]*)

Description

Returns the von Mises stress for the solid (or null if the value cannot be calculated)

Arguments

- **options (optional)** (object)

Object containing options for getting data. Can be any of:

Object has the following properties:

Name	Type	Description
ip	integer	Integration point number to get the data at (ip >= 1)

Returns

von Mises stress

Return type

real

Example

To return the von Mises stress of solid s:

```
var svm = s.VonMisesStress();
if (svm !== null) do_something...
```

Tshell class

The Tshell class gives you access to thick shell elements in D3PLOT. More...

The D3PLOT JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BlankAll(window*[GraphicsWindow]*, model*[Model]*)
- BlankFlagged(window*[GraphicsWindow]*, model*[Model]*, flag*[Flag]*)
- First(model*[Model]*)
- FlagAll(model*[Model]*, flag*[Flag]*)
- GetAll(model*[Model]*, property (optional)*[string]*)
- GetFlagged(model*[Model]*, flag*[Flag]*, property (optional)*[string]*)
- GetFromID(model*[Model]*, label*[integer]*)
- GetFromIndex(model*[Model]*, index*[integer]*)
- GetMultipleData(component*[constant]*, items*[array]*, options (optional)*[object]*)
- Last(model*[Model]*)
- Pick()
- Select(flag*[Flag]*)
- Total(model*[Model]*)
- TotalDeleted(model*[Model]*)
- UnblankAll(window*[GraphicsWindow]*, model*[Model]*)
- UnblankFlagged(window*[GraphicsWindow]*, model*[Model]*, flag*[Flag]*)
- UnflagAll(model*[Model]*, flag*[Flag]*)

Member functions

- Blank(window*[GraphicsWindow]*)
- Blanked(window*[GraphicsWindow]*)
- ClearFlag(flag*[Flag]*)
- Deleted()
- Flagged(flag*[Flag]*)
- GetData(component*[constant]*, options (optional)*[object]*)
- LocalAxes()
- Next()
- PlasticStrain(options (optional)*[object]*)
- Previous()
- SetFlag(flag*[Flag]*)
- StrainTensor(options (optional)*[object]*)
- StressTensor(options (optional)*[object]*)
- Topology()
- Unblank(window*[GraphicsWindow]*)
- VonMisesStress(options (optional)*[object]*)

Tshell properties

Name	Type	Description
colour	Colour	The colour of the tshell
data (read only)	real|array	Component data for a tshell passed as an argument to GetMultipleData. Note that data will only exist for the instance of the tshell passed to GetMultipleData. i.e. it is a local property stored on the specific instance. It is not stored in the D3PLOT database
displayMode	integer	The display mode of the tshell. Can be Constant.WIREFRAME, Constant.HIDDEN, Constant.SHADED or Constant.CURRENT.

include (read only)	integer	The include file number in the model that the tshell is in
index (read only)	integer	The internal index for the tshell in D3PLOT (starting at 0)
integrationPoints (read only)	integer	The number of through thickness integration points that the thick shell has
label (read only)	integer	The Ansys LS-DYNA label for the tshell
material (read only)	Material	The Material the tshell has. This is only available if there is a ztf file for the model. If not null will be returned. If this is a PART_COMPOSITE then null will be returned. If the model comes from writing a PTF/d3plot file from PRIMER and the material was latent there, null will be returned. Part.GetCompositeData should be used to get material data in this case
model (read only)	Model	The Model that the tshell is in
onPlanIntegrationPoints (read only)	integer	The number of on plan integration points that the thick shell has
overlayColour	Colour	The overlay colour of the tshell
overlayMode	integer	The overlay mode of the tshell. Can be Constant.NO_OVERLAY, Constant.FREE_EDGE, Constant.ALL_EDGES or Constant.CURRENT.
part (read only)	Part	The Part the tshell is in
transparency	integer	The transparency of the tshell between 0 (opaque) and 100 (clear)
type (read only)	constant	The type for the tshell (will be Type.TSHELL)

Detailed Description

The Tshell class allows you to inspect thick shell elements in a model. See the documentation below for more details.

Details of functions

Blank(window*[GraphicsWindow]*)

Description

Blanks the tshell in a graphics window

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to blank the tshell in

Returns

No return value

Example

To blank tshell t in graphics window g:

```
t.Blank(g);
```

BlankAll(window*[GraphicsWindow]*, model*[Model]*) [static]

Description

Blanks all of the tshells in the model

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to blank the tshells in

- **model** (Model)

Model that all the tshells will be blanked in

Returns

No return value

Example

To blank all of the tshells in model m, in graphics window gw:

```
Tshell.BlankAll(gw, m);
```

BlankFlagged(window*[GraphicsWindow]*, model*[Model]*, flag*[Flag]*) [static]

Description

Blanks all of the tshells in the model flagged with a defined flag

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to blank the tshells in

- **model** (Model)

Model that the flagged tshells will be blanked in

- **flag** (Flag)

Flag (see AllocateFlag) set on the tshells to blank

Returns

No return value

Example

To blank all of the tshells flagged with flag f in model m, in graphics window gw:

```
Tshell.BlankFlagged(gw, m, f);
```

Blanked(window*[GraphicsWindow]*)

Description

Checks if the tshell is blanked in a graphics window or not

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) in which to check if the tshell is blanked

Returns

true if blanked, false if not

Return type

boolean

Example

To check if tshell t is blanked in graphics window g:

```
if (t.Blanked(g) ) do_something...
```

ClearFlag(flag*[Flag]*)

Description

Clears a flag on a tshell

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to clear on the tshell

Returns

No return value

Example

To clear flag f on tshell t:

```
t.ClearFlag();
```

Deleted()

Description

Checks if the thick shell has been deleted or not

Arguments

No arguments

Returns

true if deleted, false if not

Return type

boolean

Example

To check if thick shell t has been deleted:

```
if (t.Deleted() ) do_something...
```

First(model*[Model]*) [static]

Description

Returns the first tshell in the model (or null if there are no tshells in the model)

Arguments

- **model** (Model)

Model to get first tshell in

Returns

Tshell object

Return type

Tshell

Example

To get the first tshell in model m:

```
var t = Tshell.First(m);
```

FlagAll(model*[Model]*, flag*[Flag]*) [static]

Description

Flags all of the tshells in the model with a defined flag

Arguments

- **model** (Model)

Model that all the tshells will be flagged in

- **flag** (Flag)

Flag (see AllocateFlag) to set on the tshells

Returns

No return value

Example

To flag all of the tshells with flag f in model m:

```
Tshell.FlagAll(m, f);
```

Flagged(flag*[Flag]*)

Description

Checks if the tshell is flagged or not

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to test on the tshell

Returns

true if flagged, false if not

Return type

boolean

Example

To check if tshell t has flag f set on it:

```
if (t.Flagged(f) ) do_something...
```

GetAll(model*[Model]*, property (optional)*[string]*) [static]

Description

Returns an array of Tshell objects or properties for all of the tshells in the model. If the optional property argument is not given then an array of Tshell objects is returned. If the property argument is given, that property value for each tshell is returned in the array instead of a Tshell object

Arguments

- **model** (Model)

Model that all the tshells are in

- **property (optional)** (string)

Name for property to get for all tshells in the model

Returns

Array of Tshell objects or properties

Return type

Array

Example

To get all of the tshells in model m:

```
var t = Tshell.GetAll(m);
```

To return an array containing the value of property 'foo' (for example 'label' for a tshell) for each tshell in model m:

```
var a = Tshell.GetAll(m, 'foo');
```

GetData(component*[constant]*, options (optional)*[object]*)

Description

Returns the value for a data component.
Also see GetMultipleData

Arguments

- **component** (constant)

Component constant to get data for

- **options (optional)** (object)

Object containing options for getting data. Can be any of:

Object has the following properties:

Name	Type	Description
extra	integer	The extra data component number if component Component.SOX for solids, Component.BMX for beams or Component.SHX for shells and thick shells
ip	integer	Integration point number to get the data at (ip >= 1 or one of the constants Constant.TOP, Constant.MIDDLE or Constant.BOTTOM). If the integration point is not defined it will use the integration point defined on the current GUI "data" panel, which defaults to the middle surface for shells, thick shells, and solids, and Mag All for beams, but may vary if changed by an interactive user. If consistent output from a script is required, independent of any prior interactive activity, an explicit integration point or surface should be defined
op	integer	On plane integration point number for shells and thick shells (op >= 1 [default])

referenceFrame	constant	The frame of reference to return values in. Either Constant.GLOBAL (default), Constant.LOCAL, Constant.CYLINDRICAL, Constant.USER_DEFINED or Constant.MATERIAL. This is only necessary for directional components (eg X stress) and then only when something other than the default Constant.GLOBAL coordinate system is to be used
user	integer	The user-defined component number if component Component.UNOS, Component.UNOV, Component.USSS, Component.USST, Component.UBMS or Component.UBMV

Returns

Number if a scalar component, array if a vector or tensor component (or null if the value cannot be calculated because it's not available in the model).
If requesting an invalid component it will throw an error (e.g. Component.AREA of a node).

Return type

real|array

Example

To calculate a component and check it has been calculated (note that in the example, the argument extra is optional):

```
var value = t.GetData(component, {extra: 1});
if (value !== null) do_something...
```

GetFlagged(model*[Model]*, flag*[Flag]*, property (optional)*[string]*) [static]

Description

Gets all of the tshells in the model flagged with a defined flag. If the optional property argument is not given then an array of Tshell objects is returned. If the property argument is given, that property value for each tshell is returned in the array instead of a Tshell object

Arguments

- **model** (Model)

Model that the flagged tshells are in

- **flag** (Flag)

Flag (see AllocateFlag) set on the tshells to get

- **property (optional)** (string)

Name for property to get for all flagged tshells in the model

Returns

Array of Tshell objects or properties

Return type

Array

Example

To get all of the tshells flagged with flag f in model m:

```
Tshell.GetFlagged(m, f);
```

To return an array containing the value of property 'foo' (for example 'label' for a node) for all tshells flagged with flag f in model m:

```
var a = Tshell.GetFlagged(m, f, 'foo');
```

GetFromID(model*[Model]*, label*[integer]*) [static]

Description

Returns the Tshell object for tshell in model with label (or null if it does not exist)

Arguments

- **model** (Model)

Model to get tshell in

- **label** (integer)

The Ansys LS-DYNA label for the tshell in the model

Returns

Tshell object

Return type

Tshell

Example

To get the tshell in model m with label 1000:

```
var t = Tshell.GetFromID(m, 1000);
```

GetFromIndex(model*[Model]*, index*[integer]*) [static]

Description

Returns the Tshell object for tshell in model with index (or null if it does not exist)

Arguments

- **model** (Model)

Model to get tshell in

- **index** (integer)

The D3PLOT internal index in the model for tshell, starting at 0.

Returns

Tshell object

Return type

Tshell

Example

To get the 51st tshell in model m:

```
var t = Tshell.GetFromIndex(m, 50);
```

GetMultipleData(component*[constant]*, items*[array]*, options (optional)*[object]*) [static]

Description

Returns the value for a data component for multiple tshells. For each tshell a local property called data will be created containing a number if a scalar component, or an array if a vector or tensor component (or null if the value cannot be calculated). The data is also returned as an object.
Also see GetData

Arguments

- **component** (constant)

Component constant to get data for

- **items** (array)

Array of Tshell objects to get the data for. All of the tshells must be in the same model.

- **options (optional)** (object)

Object containing options for getting data. Can be any of:

Object has the following properties:

Name	Type	Description
extra	integer	The extra data component number if component Component.SOX for solids, Component.BMX for beams or Component.SHX for shells and thick shells
ip	integer	Integration point number to get the data at (ip >= 1 or one of the constants Constant.TOP, Constant.MIDDLE or Constant.BOTTOM)
op	integer	On plane integration point number for shells and thick shells (op >= 1 [default])
referenceFrame	constant	The frame of reference to return values in. Either Constant.GLOBAL (default), Constant.LOCAL, Constant.CYLINDRICAL, Constant.USER_DEFINED or Constant.MATERIAL. This is only necessary for directional components (eg X stress) and then only when something other than the default Constant.GLOBAL coordinate system is to be used
user	integer	The user-defined component number if component Component.UNOS, Component.UNOV, Component.USSS, Component.USST, Component.UBMS or Component.UBMV

Returns

Object containing the data. A property is created in the object for each tshell with the label. The value of the property is a number if a scalar component or an array if a vector or tensor component (or null if the value cannot be calculated)

Return type

object

Example

To calculate a component for tshells in array items and use the data property (note that in the example, the argument extra is optional):

```
Tshell.GetMultipleData(component, items, {extra: 1});
for (i=0; i<items.length; i++)
{
    if (items[i].data !== null) do_something...
}
```

To calculate a component for tshells in array items and use the return value (note that in the example, the argument extra is optional):

```
var data = Tshell.GetMultipleData(component, items, {extra: 1});
for (d in data)
{
    Message("Label is " + d);
    if (data[d] !== null) do_something...
}
```

Last(model[*Model*]) [static]

Description

Returns the last tshell in the model (or null if there are no tshells in the model)

Arguments

- **model** (Model)

Model to get last tshell in

Returns

Tshell object

Return type

Tshell

Example

To get the last tshell in model m:

```
var t = Tshell.Last(m);
```

LocalAxes()

Description

Returns the local axes of the element in model space, expressed as direction cosines in a 2D array. Beam elements must have 3 nodes to be able to return local axes.

Arguments

No arguments

Returns

array of arrays

Return type

Array

Example

To get the local axes for tshell t:

```
var axes = t.LocalAxes();
var xAxis = [ axes[0][0], axes[0][1], axes[0][2] ];
var yAxis = [ axes[1][0], axes[1][1], axes[1][2] ];
var zAxis = [ axes[2][0], axes[2][1], axes[2][2] ];
```

Next()

Description

Returns the next tshell in the model (or null if there is not one)

Arguments

No arguments

Returns

Tshell object

Return type

Tshell

Example

To get the next tshell after tshell t:

```
t = t.Next();
```

Pick() [static]

Description

Allows the user to pick a tshell from the screen

Arguments

No arguments

Returns

Tshell object or null if cancelled

Return type

Tshell

Example

To pick a tshell:

```
var t = Tshell.Pick();
```

PlasticStrain(options (optional)*[object]*)

Description

Returns the effective plastic strain for the thick shell (or null if the value cannot be calculated)

Arguments

- **options (optional)** (object)

Object containing options for getting data. Can be any of:

Object has the following properties:

Name	Type	Description
ip	integer	Integration point number to get the data at (ip >= 1 or one of the constants Constant.TOP, Constant.MIDDLE or Constant.BOTTOM)
op	integer	On plane integration point number for shells and thick shells (op >= 1 [default])

Returns

Plastic strain

Return type

real

Example

To return the effective plastic strain of thick shell t:

```
var strain = t.PlasticStrain();
if (strain !== null) do_something...
```

Previous()

Description

Returns the previous tshell in the model (or null if there is not one)

Arguments

No arguments

Returns

Tshell object

Return type

Tshell

Example

To get the previous tshell before tshell t:

```
t = t.Previous();
```

Select(flag*[Flag]*) [static]

Description

Selects tshells using an object menu

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to use when selecting tshells

Returns

The number of tshells selected or null if menu cancelled

Return type

integer

Example

To select tshells, flagging those selected with flag f:

```
var total = Tshell.Select(f);
```

SetFlag(flag*[Flag]*)

Description

Sets a flag on a tshell

Arguments

- **flag** (Flag)

Flag (see AllocateFlag) to set on the tshell

Returns

No return value

Example

To set flag f on tshell t:

```
t.SetFlag(f);
```

StrainTensor(options (optional)*[object]*)

Description

Returns the strain tensor for the thick shell

Arguments

- **options (optional)** (object)

Object containing options for getting data. Can be any of:

Object has the following properties:

Name	Type	Description
ip	integer	Integration point number to get the data at (ip >= 1 or one of the constants Constant.TOP, Constant.MIDDLE or Constant.BOTTOM)
op	integer	On plane integration point number for shells and thick shells (op >= 1 [default])
referenceFrame	constant	The frame of reference to return values in. Either Constant.GLOBAL (default), Constant.LOCAL, Constant.CYLINDRICAL, Constant.USER_DEFINED or Constant.MATERIAL

Returns

Array containing the strain tensor [Exx, Eyy, Ezz, Exy, Eyz, Ezx] (or null if the value cannot be calculated)

Return type

array

Example

To return the strain tensor of think shell t:

```
var tensor = t.StrainTensor();
if (tensor !== null) do_something...
```

StressTensor(options (optional)*[object]*)

Description

Returns the stress tensor for the thick shell

Arguments

- **options (optional)** (object)

Object containing options for getting data. Can be any of:

Object has the following properties:

Name	Type	Description
ip	integer	Integration point number to get the data at (ip >= 1 or one of the constants Constant.TOP, Constant.MIDDLE or Constant.BOTTOM)
op	integer	On plane integration point number for shells and thick shells (op >= 1 [default])

referenceFrame	constant	The frame of reference to return values in. Either Constant.GLOBAL (default), Constant.LOCAL, Constant.CYLINDRICAL, Constant.USER_DEFINED or Constant.MATERIAL

Returns

Array containing the stress tensor [Exx, Eyy, Ezz, Exy, Eyz, Ezx] (or null if the value cannot be calculated)

Return type

array

Example

To return the stress tensor of thick shell t:

```
var tensor = t.StressTensor();
if (tensor !== null) do_something...
```

Topology()

Description

Returns the topology for the tshell in the model

Arguments

No arguments

Returns

array of Node objects

Return type

Array

Example

To get the topology for tshell t:

```
var topology = t.Topology();
```

Total(model[*Model*]) [static]

Description

Returns the total number of tshells in the model

Arguments

- **model** (Model)

Model to get total in

Returns

The number of tshells

Return type

integer

Example

To get the number of tshells in model m:

```
var total = Tshell.Total(m);
```

TotalDeleted(model*[Model]*) [static]

Description

Returns the total number of thick shells that have been deleted in a model in the state given by its state property

Arguments

- **model** (Model)

Model to get total in

Returns

The number of thick shells that have been deleted

Return type

integer

Example

To get the number of thick shells in model m that have been deleted in state 10:

```
m.state = 10;
var total = Tshell.TotalDeleted(m);
```

Unblank(window*[GraphicsWindow]*)

Description

Unblanks the tshell in a graphics window

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to unblank the tshell in

Returns

No return value

Example

To unblank tshell t in graphics window g:

```
t.Unblank(g);
```

UnblankAll(window*[GraphicsWindow]*, model*[Model]*) [static]

Description

Unblanks all of the tshells in the model

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to unblank the tshells in

- **model** (Model)

Model that all the tshells will be unblanked in

Returns

No return value

Example

To unblank all of the tshells in model m, in graphics window gw:

```
Tshell.UnblankAll(gw, m);
```

UnblankFlagged(window*[GraphicsWindow]*, model*[Model]*, flag*[Flag]*) [static]

Description

Unblanks all of the tshells in the model flagged with a defined flag

Arguments

- **window** (GraphicsWindow)

GraphicsWindow) to unblank the tshells in

- **model** (Model)

Model that the flagged tshells will be unblanked in

- **flag** (Flag)

Flag (see AllocateFlag) set on the tshells to unblank

Returns

No return value

Example

To unblank all of the tshells flagged with flag f in model m, in graphics window gw:

```
Tshell.UnblankFlagged(gw, m, f);
```

UnflagAll(model*[Model]*, flag*[Flag]*) [static]

Description

Unsets a defined flag on all of the tshells in the model

Arguments

- **model** (Model)

Model that the defined flag for all tshells will be unset in

- **flag** (Flag)

Flag (see AllocateFlag) to unset on the tshells

Returns

No return value

Example

To unset flag f on all of the tshells in model m:

```
Tshell.UnflagAll(m, f);
```

VonMisesStress(options (optional)*[object]*)

Description

Returns the von Mises stress for the thick shell (or null if the value cannot be calculated)

Arguments

- **options (optional)** (object)

Object containing options for getting data. Can be any of:

Object has the following properties:

Name	Type	Description
ip	integer	Integration point number to get the data at (ip >= 1 or one of the constants Constant.TOP, Constant.MIDDLE or Constant.BOTTOM)
op	integer	On plane integration point number for shells and thick shells (op >= 1 [default])

Returns

von Mises stress

Return type

real

Example

To return the von Mises stress of thick shell t:

```
var svm = t.VonMisesStress();
if (svm !== null) do_something...
```

Type class

The Type class defines constants for the types in D3PLOT. More...

The D3PLOT JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Type constants

Name	Description
Type.BEAM	Beam elements
Type.BOLT	Bolts (contributes to the count of total connections, but can't provide data or be visualised)
Type.BWLD	Beam spotwelds
Type.CONTACT	Contact surfaces
Type.CONX	All connection types
Type.CWLD	*CONSTRAINED_SPOTWELD spotwelds
Type.DES	Discrete Element Sphere
Type.ELEMENT	Generic elements
Type.GROUP	Groups
Type.GWLD	*CONSTRAINED_GENERALIZED spotwelds
Type.HSWA	Hex spotweld assemblies
Type.HWLD	Hex (Solid) spotwelds
Type.JOINT	Joints
Type.MASS	Lumped masses
Type.MATERIAL	Materials
Type.MIG	MIG welds (contributes to the count of total connections, but can't provide data or be visualised)
Type.MODEL	Model
Type.NODE	Nodes
Type.NRB	Nodal Rigid Bodies
Type.PART	Parts
Type.PRETENSIONER	Pretensioners
Type.RBOLT	Rigid bolts (contributes to the count of total connections, but can't provide data or be visualised)
Type.RETRACTOR	Retractors
Type.RIGIDWALL	Rigidwalls
Type.SBENT	Seatbelt types generally

Type.SEATBELT	Seatbelt elements
Type.SECTION	(Element) section definitions
Type.SEGMENT	Contact segments
Type.SET_BEAM	*SET_BEAM sets
Type.SET_DISCRETE	*SET_DISCRETE sets
Type.SET_NODE	*SET_NODE sets
Type.SET_PART	*SET_PART sets
Type.SET_SHELL	*SET_SHELL sets
Type.SET_SOLID	*SET_SOLID sets
Type.SET_TSHELL	*SET_TSHELL sets
Type.SHELL	Shell elements
Type.SLIPRING	Slip-rings
Type.SOLID	Solid elements
Type.SPC	Single Point Constraint
Type.SPH	Smoothed Particle Hydrodynamics
Type.SPRING	Springs (discrete elements)
Type.TSHELL	Thick shell elements
Type.WINDOW	D3PLOT window id
Type.XSEC	Database cross-sections

Detailed Description

The Type class gives you access to the different types that are used in the D3PLOT API. See the documentation below for more details.

View class

The View class allows you to control the view and plotting modes in D3PLOT. More...

The D3PLOT JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Ac()
- Ct()
- Hi()
- Li()
- Redraw()
- Sh()
- Show(View type*[constant]*)
- Si()
- Vec()

View constants

Constants for Show

Name	Description
View.ISO	Isometric projection
View.XY	XY axis projection
View.XZ	XZ axis projection
View.YZ	YZ axis projection

Constants for plotting mode

Name	Description
View.CURRENT	current mode
View.HIDDEN	Hidden line
View.SHADED	Shaded view
View.WIRE	Wireframe

Detailed Description

The View class gives you access to the different plotting modes and views. See the documentation below for more details.

Details of functions

Ac() [static]

Description

Autoscales the view

Arguments

No arguments

Returns

No return value

Example

To autoscale

```
View.Ac();
```

Ct() [static]

Description

Does a contour plot

Arguments

No arguments

Returns

No return value

Example

To do a contour plot

```
View.Ct();
```

Hi() [static]

Description

Does a Hidden line plot

Arguments

No arguments

Returns

No return value

Example

To do a hidden line plot

```
View.Hi();
```

Li() [static]

Description

Does a line (wireframe) plot

Arguments

No arguments

Returns

No return value

Example

To do a line plot

```
View.Li();
```

Redraw() [static]

Description

Redraws the plot using the current plot mode.

Arguments

No arguments

Returns

No return value

Example

To redraw

```
View.Redraw();
```

Sh() [static]

Description

Does a shaded plot

Arguments

No arguments

Returns

No return value

Example

To do a shaded plot

```
View.Sh();
```

Show(View type*[constant]*) [static]

Description

Redraws using one of the standard views

Arguments

- **View type** (constant)

The view to show. Can be +/-View.XY, +/-View.YZ, +/-View.XZ or +/-View.ISO

Returns

No return value

Example

To do an isometric view from the negative direction:

```
View.Show(-View.ISO);
```

Si() [static]

Description

Does a shaded image contour plot

Arguments

No arguments

Returns

No return value

Example

To do a shaded image contour plot

```
View.Si();
```

Vec() [static]

Description

Does a vector plot

Arguments

No arguments

Returns

No return value

Example

To do a vector plot

```
View.Vec();
```

Widget class

The Widget class allows you to create components for a graphical user interface. More...

The D3PLOT JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- CtrlPressed()
- PixelsPerUnit()
- ShiftPressed()
- StringLength(text*[string]*, monospace (optional)*[boolean]*, fontSize (optional)*[integer]*)

Member functions

- AddWidgetItem(item*[WidgetItem]*, position (optional)*[integer]*)
- AddWidgetItem(item*[WidgetItem]*, relationship*[constant]*, relitem*[WidgetItem]*)
- Circle(colour*[constant]*, fill*[boolean]*, xc*[integer]*, yc*[integer]*, radius*[integer]*)
- Clear()
- ClearSelection()
- Cross(colour (optional)*[constant]*)
- Delete()
- DirectoryIcon(line_colour*[constant]*, fill_colour*[constant]*)
- DumpImageString(filename*[string]*, format (optional)*[constant]*)
- Hide()
- ItemAt(index*[integer]*)
- Line(colour*[constant]*, x1*[integer]*, y1*[integer]*, x2*[integer]*, y2*[integer]*)
- MoveWidgetItem(item*[WidgetItem]*, relationship*[constant]*, relitem*[WidgetItem or null]*)
- Polygon(colour*[constant]*, fill*[boolean]*, points*[Array of numbers]*)
- Polygon(colour*[constant]*, fill*[boolean]*, x1*[integer]*, y1*[integer]*, x2*[integer]*, y2*[integer]*, ... xn*[integer]*, ... yn*[integer]*) [deprecated]
- ReadImageFile(filename*[string]*, justify (optional)*[constant]*, transparent (optional)*[colour value (integer)]*, tolerance (optional)*[integer]*)
- ReadImageString(string*[string]*, justify (optional)*[constant]*, transparent (optional)*[colour value (integer)]*, tolerance (optional)*[integer]*)
- Rectangle(colour*[constant]*, fill*[boolean]*, x1*[integer]*, y1*[integer]*, x2*[integer]*, y2*[integer]*)
- RemoveAllWidgetItems()
- RemoveWidgetItem(item*[WidgetItem]*)
- Scroll(scroll*[constant or WidgetItem object]*)
- Show()
- Static()
- Tick(colour (optional)*[constant]*)
- TotalItems()
- WidgetItems()

Widget constants

Name	Description
Widget.BUTTON	Button widget
Widget.CHECKBOX	Checkbox widget
Widget.COMBOBOX	Combobox widget
Widget.LABEL	Label widget
Widget.LISTBOX	Listbox widget

Widget.RADIOBUTTON	Radiobutton widget
Widget.SLIDER	Slider widget
Widget.TEXTBOX	Text input widget
Widget.TREE	Tree widget

Constants for Colour

Name	Description
Widget.BLACK	Colour black
Widget.BLUE	Colour blue
Widget.COLOUR_CONTRAST	A contrasting colour in the 3 user interface themes (Green, Purple, and Blue in the Dark, Light, and Classic themes respectively). Blue in the legacy theme.
Widget.COLOUR_CONTRAST_2	Another contrasting colour in the 3 user interface themes (Yellow, Red, and Red in the Dark, Light, and Classic themes respectively). Red in the legacy theme.
Widget.COLOUR_INVERSE	Inverse colour in the 3 user interface themes (Black or white depending on theme). Black in the legacy theme.
Widget.COLOUR_LABEL	Label text colour in the 3 user interface themes (Black or white depending on theme). Black in the legacy theme.
Widget.COLOUR_LATENT	Latent colour in the 3 user interface themes (Different shade of Cyan in every theme). Light Cyan in the legacy theme.
Widget.COLOUR_NEUTRAL	Neutral colour in the 3 user interface themes (Different shade of grey in every theme). Light grey in the legacy theme.
Widget.COLOUR_SAFE	Safe colour in the 3 user interface themes (Different shade of green in every theme). Dark green in the legacy theme.
Widget.COLOUR_TITLE	Title colour in the 3 user interface themes (Different shade of grey in every theme). Dark blue in the legacy theme.
Widget.COLOUR_WARNING	Warning colour in the 3 user interface themes (Different shade of red in every theme). Dark red in the legacy theme.
Widget.CYAN	Colour cyan
Widget.DARKBLUE	Colour dark blue
Widget.DARKGREEN	Colour dark green
Widget.DARKGREY	Colour dark grey
Widget.DARKGREY_NEUTRAL	Only valid in the function 'Line'. Used to keep the 3D effect in the legacy theme and not in the other themes. Neutral colour in the 3 user interface themes (Different shade of grey in every theme). Dark grey in the legacy theme
Widget.DARKRED	Colour dark red
Widget.DEFAULT	Default colour for widgets
Widget.GREEN	Colour green
Widget.GREY	Colour grey
Widget.LIGHTGREY	Colour light grey
Widget.LIGHTGREY_NEUTRAL	Only valid in the function 'Line'. Used to keep the 3D effect in the legacy theme and not in the other themes. Neutral colour in the 3 user interface themes (Different shade of grey in every theme). Light grey in the legacy theme
Widget.MAGENTA	Colour magenta
Widget.ORANGE	Colour orange

Widget.RED	Colour red
Widget.WHITE	Colour white
Widget.YELLOW	Colour yellow

Constants for Image RGB format

Name	Description
Widget.RGB24	24 bits for RGB data in widget images
Widget.RGB8	8 bits for RGB data in widget images

Constants for Justification

Name	Description
Widget.BOTTOM	Bottom justification
Widget.CENTRE	Centre (horizontal) justification
Widget.LEFT	Left justification
Widget.MIDDLE	Middle (vertical) justification
Widget.RIGHT	Right justification
Widget.SCALE	Image will be scaled to fit widget
Widget.TOP	Top justification

Constants for Orientation

Name	Description
Widget.HORIZONTAL	Horizontal orientation (for sliders)
Widget.VERTICAL	Vertical orientation (for sliders)

Constants for Selection

Name	Description
Widget.SELECT_ENHANCED	Multiple WidgetItems in a ListBox or tree Widget can be selected. When the user selects a WidgetItem the selection is cleared and the new WidgetItem selected. However, if the user presses the Ctrl key when clicking on a WidgetItem, the clicked WidgetItem gets toggled and all other WidgetItems are left untouched. If the user presses the Shift key while clicking on a WidgetItem, all WidgetItems between the last selected WidgetItem and the clicked WidgetItem are selected or unselected, depending on the state of the clicked WidgetItem.
Widget.SELECT_MULTIPLE	Multiple WidgetItems in a ListBox Widget can be selected. When the user selects a WidgetItem, the selection status of that WidgetItem is toggled and the other WidgetItems are left alone. Not valid for tree widgets. Widget.SELECT_ENHANCED will be used.
Widget.SELECT_NONE	No WidgetItem in a ListBox or tree Widget can be selected
Widget.SELECT_SINGLE	A single WidgetItem in a ListBox or tree Widget can be selected. When the user selects a WidgetItem, any already-selected WidgetItem becomes unselected, and the user cannot unselect the selected WidgetItem by clicking on it.

Constants for Tree relations

Name	Description

Widget.AFTER	Add a WidgetItem after the existing WidgetItem for a tree widget.
Widget.BEFORE	Add a WidgetItem before the existing WidgetItem for a tree widget.
Widget.CHILD	Add a WidgetItem as a child of the existing WidgetItem for a tree widget.

Constants for Tree/listbox scrolling

Name	Description
Widget.SCROLL_BOTTOM	Scroll tree or listbox widget to bottom.
Widget.SCROLL_DOWN	Scroll tree or listbox widget down one.
Widget.SCROLL_PAGE_DOWN	Scroll tree or listbox widget down one page.
Widget.SCROLL_PAGE_UP	Scroll tree or listbox widget up one page.
Widget.SCROLL_TOP	Scroll tree or listbox widget to top.
Widget.SCROLL_UP	Scroll tree or listbox widget up one.

Constants for User interface categories

Name	Description
Widget.CATEGORY_APPLY	Apply buttons
Widget.CATEGORY_BUTTON_BOX	A button box panel that contains other widgets
Widget.CATEGORY_CANCEL	Buttons which cancel the current operation
Widget.CATEGORY_DATA_ ENTRY_HEADER	Header for data entry cells, e.g. PRIMER create panels
Widget.CATEGORY_DISMISS	Buttons to close or dismiss panels
Widget.CATEGORY_ENTITY	Entity types in T/HIS
Widget.CATEGORY_GENERIC	A generic button that isn’t a special category
Widget.CATEGORY_GENERIC_2	An alternative to the generic category that has a complementary colour
Widget.CATEGORY_HELP	Help buttons
Widget.CATEGORY_KEYWORD	A PRIMER keyword button
Widget.CATEGORY_LABEL	A text label
Widget.CATEGORY_LABEL_BOX	Text label with a border
Widget.CATEGORY_LABEL_ POPUP	Text label with a popup that blends into the background
Widget.CATEGORY_MENU_BOX	A menu box
Widget.CATEGORY_MESSAGE	For displaying a temporary warning message
Widget.CATEGORY_OPERATE	Operate buttons in T/HIS
Widget.CATEGORY_POPUP_BOX	A popup box that can contain buttons and plain text
Widget.CATEGORY_SAFE_ ACTION	Buttons (usually green) to indicate a safe action
Widget.CATEGORY_SEL_ALL	Select all

Widget.CATEGORY_TAB	Tab
Widget.CATEGORY_TABLE_HEADER	Table (column) header
Widget.CATEGORY_TABLE_ROW	Table row
Widget.CATEGORY_TEXT_BOX	A text box
Widget.CATEGORY_TICKBOX	A tick box
Widget.CATEGORY_TITLE	Title text
Widget.CATEGORY_TOGGLE	Buttons that can be toggled, e.g. On/Off
Widget.CATEGORY_TOOL	Buttons within the tools area
Widget.CATEGORY_UNDO	Buttons which undo the last operation
Widget.CATEGORY_UNSEL_ALL	Unselect/deslect all
Widget.CATEGORY_UPDATE	Update buttons which update the screen but leave the panel open
Widget.CATEGORY_WARNING_ACTION	Buttons (usually red) to indicate a dangerous action
Widget.NO_CATEGORY	No styling is applied. Widget colour controlled by foreground/background properties and is the same in all themes

Widget properties

Name	Type	Description
active	logical	If widget is active (true) or disabled (false)
arrows	boolean	Whether arrows will be shown for a slider (default is true). Slider Widgets only.
background	constant	Widget background colour. Can be: Widget.BLACK, Widget.WHITE, Widget.RED, Widget.GREEN, Widget.BLUE, Widget.CYAN, Widget.MAGENTA, Widget.YELLOW, Widget.DARKRED, Widget.DARKGREEN, Widget.DARKBLUE, Widget.GREY, Widget.DARKGREY, Widget.LIGHTGREY, Widget.ORANGE, Widget.DEFAULT, Widget.COLOUR_NEUTRAL, Widget.COLOUR_CONTRAST, Widget.COLOUR_CONTRAST_2, Widget.COLOUR_WARNING, Widget.COLOUR_SAFE, Widget.COLOUR_TITLE, Widget.COLOUR_INVERSE, Widget.DARKGREY_NEUTRAL, Widget.LIGHTGREY_NEUTRAL Widget.COLOUR_LATENT, Note, background colours in the Window.THEME_DARK, Window.THEME_LIGHT, and Window.THEME_CLASSIC themes will be determined by the category of the widget not the background colour. To override this behaviour and use this background colour first set the widget category to Widget.NO_CATEGORY.
bottom	integer	Widget bottom coordinate
category	constant	The button category which determines the button's appearance when using the new user interface, see Window.Theme()
currentItem	WidgetItem object	The current WidgetItem for a tree Widget. The current WidgetItem in a tree is shown with a dashed border.
fontSize	integer	Widget font size in points. Currently only supports the following sizes: 6, 7, 8, 10, 12, 14, 18, 24. Can be used only with Widget.LABEL and Widget.BUTTON. Both LATIN1 and UTF-8 encoding is supported on Windows but Linux only supports LATIN1 encoding at the moment.

foreground	constant	Widget foreground colour. Can be: Widget.BLACK, Widget.WHITE, Widget.RED, Widget.GREEN, Widget.BLUE, Widget.CYAN, Widget.MAGENTA, Widget.YELLOW, Widget.DARKRED, Widget.DARKGREEN, Widget.DARKBLUE, Widget.GREY, Widget.DARKGREY, Widget.LIGHTGREY, Widget.ORANGE, Widget.DEFAULT, Widget.COLOUR_NEUTRAL, Widget.COLOUR_CONTRAST, Widget.COLOUR_CONTRAST_2, Widget.COLOUR_WARNING, Widget.COLOUR_SAFE, Widget.COLOUR_TITLE, Widget.COLOUR_LABEL, Widget.COLOUR_INVERSE, Widget.DARKGREY_NEUTRAL, Widget.LIGHTGREY_NEUTRAL Widget.COLOUR_LATENT, Note, foreground colours in the Window.THEME_DARK, Window.THEME_LIGHT, and Window.THEME_CLASSIC themes will be determined by the category of the widget not the foreground colour. To override this behaviour and use this foreground colour first set the widget category to Widget.NO_CATEGORY.
hover	string	Widget hover text
imageHeight (read only)	integer	Height of widget image (pixels)
imageWidth (read only)	integer	Width of widget image (pixels)
justify	constant	Widget justification. Can be: Widget.LEFT, Widget.RIGHT or Widget.CENTRE (default).
left	integer	Widget left coordinate
lineWidth	integer	Width of lines when drawing graphics (initially 1; values 1-100 allowed).
macroTag	string	Tag to use for this widget when recording a macro. If empty then the text property value will be used.
maximum	integer	The maximum value allowed for a slider (default is 100). Slider Widgets only.
minimum	integer	The minimum value allowed for a slider (default is 0). Slider Widgets only.
monospace	boolean	true if the widget uses a monospace font instead of a proportional width font (default). Label and button Widgets only.
onChange	function	Function to call when the text in a TEXTBOX widget, the selection in a COMBOBOX widget or the value of a SLIDER is changed. The Widget object is accessible in the function using the 'this' keyword (see the example below for more details of how to define the function and how to use the 'this' keyword). To unset the function set the property to null. **Note that this function is called when the user actually types something into the textbox, selects an item in the combobox or moves the slider, NOT when the Widget.text or Widget.value property changes**.
onClick	function	Function to call when a BUTTON, CHECKBOX, COMBOBOX, LABEL, RADIOBUTTON or TREE widget is clicked. The Widget object is accessible in the function using the 'this' keyword (see the example below for more details of how to define the function and how to use the 'this' keyword). To unset the function set the property to null. **Note that this function is called when the user actually clicks on the button, NOT when the Widget.pushed property changes**. For the COMBOBOX widget the function is called **before** the list of items is mapped.
onPopup	function	Function to call when a BUTTON, LABEL, TEXTBOX or TREE widget is right clicked to map a popup. The Widget object is accessible in the function using the 'this' keyword. The PopupWindow can then be found by using the popupWindow property of the Widget. The function is called **before** the popup is mapped so you can change the widgets in the popup as required.
onTimer	function	Function to call for a widget when timerDelay ms have elapsed after setting this. Additionally if timerRepeat is set this function will be called repetitively, every timerDelay ms. The Widget object is accessible in the function using the 'this' keyword. To unset the function set the property to null. **Note that as soon as this property is set the timer starts!**

orientation	constant	The orientation of a slider. Can be: Widget.VERTICAL or Widget.HORIZONTAL (default). Slider Widgets only.
popupDirection	constant	How PopupWindow will be mapped relative to this widget. Can be Widget.LEFT, Widget.RIGHT, Widget.TOP or Widget.BOTTOM (default). For tree widgets this will be ignored as the popup is always shown on the WidgetItem that is right clicked.
popupSymbol	logical	TRUE (default) if a symbol will be shown for a PopupWindow.
popupWindow	PopupWindow object	PopupWindow for this Widget. Only available for Button, Label and Textbox Widgets. To remove a PopupWindow from a Widget set to null.
pushed	logical	If widget is pushed (true) or not (false). This only affects Widget.BUTTON with the Widget.toggle property set, and Widget.CHECKBOX widgets.
right	integer	Widget right coordinate
select	constant	Selection method for ListBox and tree Widgets. Can be: Widget.SELECT_NONE, Widget.SELECT_SINGLE or Widget.SELECT_MULTIPLE or Widget.SELECT_ENHANCED (default).
selectedItem	WidgetItem object	WidgetItem that is currently selected for a ComboBox or Radiobutton, Widget. If null no WidgetItem is selected. For a ListBox Widget this property contains the last WidgetItem that was (de)selected. To get a list of all of the selected WidgetItems use WidgetItems() to return all of the WidgetItems and inspect the WidgetItem selected property.
shown (read only)	boolean	true if the widget is visible. To alter the visibility of a widget use the Show() and Hide() methods.
step	integer	The step value of a slider (default is 1). Slider Widgets only.
text	string	Widget text. For a ComboBox Widget this will be the text for the currently selected WidgetItem
textHidden	boolean	true if the widget text is hidden and replaced by asterisks. This may be used to create textboxes to type passwords in. TextBox Widgets only.
timerDelay	integer	Delay in ms before the function set for onTimer will be called. The initial value is 1000 (ms). Also see timerRepeat.
timerRepeat	logical	If the function set for onTimer will be called once (false) or repeatedly (true). The initial value is false. Also see timerDelay.
toggle	logical	If widget can be toggled (true) or not (false). This only affects Widget.BUTTON widgets.
top	integer	Widget top coordinate
type (read only)	integer	Type of the widget. The widget type could be Widget.BUTTON, Widget.CHECKBOX, Widget.COMBOBOX, Widget.LABEL, Widget.LISTBOX, Widget.RADIOBUTTON, Widget.SLIDER, Widget.TEXTBOX or Widget.TREE
value	integer	The current value of a slider (initially will be the minimum value). Slider Widgets only.
window (read only)	Window object	The Window that this widget is defined in
xResolution	integer	X resolution of button when drawing lines, circles, polygons and rectangles (initially 100). X coordinates on the Widget can be from 0 (on the left of the widget) to xResolution (on the right of the widget). Available for Widget.LABEL and Widget.BUTTON Widgets.
yResolution	integer	Y resolution of button when drawing lines, circles, polygons and rectangles (initially 100). Y coordinates on the Widget can be from 0 (on the top of the widget) to yResolution (on the bottom of the widget). Available for Widget.LABEL and Widget.BUTTON Widgets.

Detailed Description

The Widget class allows you to create Widgets (buttons, textboxes etc) in a Window for a graphical user interface. Callback functions can be declared for widgets to give actions when a button is pressed or the text in a textbox is selected etc. The following example displays various widgets in a window. Several callback methods are used. The exit button allows the user to exit the script but the button is only made active if the checkbox widget is ticked. If the button widgets are pressed feedback is given to the user

```
var count = 0;
// Create window
var w = new Window("Test", 0.8, 1.0, 0.5, 0.6);
// Create all of the widgets
var l = new Widget(w, Widget.LABEL,    1, 30, 1, 7, "Text:");
var t = new Widget(w, Widget.TEXTBOX, 31, 80, 1, 7, "Enter text");
var b = new Widget(w, Widget.BUTTON,   1, 30, 8, 14, "Press me");
var b2= new Widget(w, Widget.BUTTON,  31, 61, 8, 14, "Don't press me");
var c = new Widget(w, Widget.CHECKBOX,62, 68, 8, 14);
var l2= new Widget(w, Widget.LABEL,    1, 80, 15, 21, "You haven't pressed the
button yet...");
var e = new Widget(w, Widget.BUTTON,   1, 21, 22, 28, "Exit");
// Allow button widget b2 to toggle
b2.toggle = true;
// The exit button is initially inactive
e.active = false;
// Assign the callback functions
b.onClick  = clicked;
b2.onClick = clicked;
c.onClick  = clicked;
t.onChange = changed;
e.onClick  = confirm_exit;
// Show the window and start event loop
w.Show();
////////////////////////////////////////////////////////////////////////////
function clicked()
{
// If checkbox is clicked then set the state of the exit button
    if (this === c)
    {
        Message("Checkbox clicked");
        e.active = c.pushed;
    }
// If the "Don't press me' button is pressed then change the colour if the
button is pressed in.
    else if (this === b2)
    {
        Message("I said don't press!!!");
        if (b2.pushed) b2.background = Widget.WHITE;
        else           b2.background = Widget.DEFAULT;
    }
// If the "Press me" button is pressed then update the text in the label widget
// with how many times the button has been pressed.
    else
    {
        Message("You pressed...");
        count++;
        l2.text = "Button pressed " + count + " times";
    }
}
////////////////////////////////////////////////////////////////////////////
function changed()
{
// If the user has changed the text in the textbox then give a message in
// the dialogue box
    Message("Text has changed to " + this.text);
}
////////////////////////////////////////////////////////////////////////////
function confirm_exit()
{
// Map confirm box
    var ret = Window.Question("Confirm exit", "Are you sure you want to quit?");
```

```
// If the user has answered yes then exit from the script.
    if (ret == Window.YES) Exit();
}
```

In version 20 a tree widget was added. A simple tree widget example is shown below.

```
Window.Theme(Window.THEME_CURRENT);
let wi, cwi;
// Create a popup window and some widgets
let pw = new PopupWindow();
let pw_l1 = new Widget(pw, Widget.LABEL, 1, 61,  1,  7, "");
let pw_l2 = new Widget(pw, Widget.LABEL, 1, 61,  7, 13, "");
let pw_l3 = new Widget(pw, Widget.LABEL, 1, 61, 13, 19, "");
let pw_l4 = new Widget(pw, Widget.LABEL, 1, 61, 19, 25, "");
let pw_l5 = new Widget(pw, Widget.LABEL, 1, 61, 25, 31, "");
let pw_l6 = new Widget(pw, Widget.LABEL, 1, 61, 31, 37, "");
// Create window
let w = new Window("JavaScript Tree widget test", 0.85, 1.0, 0.75, 1.0);
// Create tree widget
let t = new Widget(w, Widget.TREE, 1, 61, 1, 51, "Suite");
// Add a root node to tree
let env_wi = new WidgetItem(t, "MyItem");
// Add a child to the root node
wi  = new WidgetItem(t, "PRIMER", Widget.CHILD, env_wi);
cwi = new WidgetItem(t, "Prepare", Widget.CHILD, wi);
wi.onMouseOver = wi_onmouseover;
cwi.hover = "Efficient, reliable model setup with support for all of the latest
Ansys LS-DYNA features";
wi  = new WidgetItem(t, "Ansys LS-DYNA", Widget.CHILD, env_wi);
cwi = new WidgetItem(t, "Analyse", Widget.CHILD, wi);
wi.onMouseOver = wi_onmouseover;
// Add a sibling node after Ansys LS-DYNA
wi  = new WidgetItem(t, "REPORTER", Widget.AFTER, wi);
cwi = new WidgetItem(t, "Report", Widget.CHILD, wi);
wi.onMouseOver = wi_onmouseover;
cwi.hover = "Automatic report generation for Ansys LS-DYNA simulations";

// Add a sibling node before REPORTER
wi  = new WidgetItem(t, "T/HIS", Widget.BEFORE, wi);
cwi = new WidgetItem(t, "Process", Widget.CHILD, wi);
wi.onMouseOver = wi_onmouseover;
cwi.hover = "Plot, manipulate and process XY data from Ansys LS-DYNA";

// Alternatively, create WidgetItem without parent and add
let d3plot_wi   = new WidgetItem(null, "D3PLOT");
t.AddWidgetItem(d3plot_wi, Widget.BEFORE, wi);
cwi = new WidgetItem(null, "Visualise");
t.AddWidgetItem(cwi, Widget.CHILD, d3plot_wi);
d3plot_wi.onMouseOver = wi_onmouseover;
cwi.hover = " In-depth 3D visualisation of Ansys LS-DYNA results";
// Expand root node
env_wi.expanded = true;
// Link popup to tree widget
t.popupWindow = pw;
// Assign callbacks
t.onClick = click_tree;
t.onPopup = do_popup;
// Show the widget and start event loop
w.Show();
////////////////////////////////////////////////////////////////////////
function do_popup()
{
    pw_l1.text = this.currentItem.text;
    if (this.currentItem.selected) pw_l2.text = "Selected";
    else                           pw_l2.text = "Not selected";
    if (this.currentItem.Parent())
        pw_l3.text = "Parent: " + this.currentItem.Parent().text;
    else
        pw_l3.text = "No parent";
    if (this.currentItem.FirstChild())
        pw_l4.text = "First child: " + this.currentItem.FirstChild().text;
    else
```

```
        pw_l4.text = "No children";
    if (this.currentItem.PreviousSibling())
        pw_l5.text = "Previous: " + this.currentItem.PreviousSibling().text;
    else
        pw_l5.text = "No previous";
    if (this.currentItem.NextSibling())
        pw_l6.text = "Next: " + this.currentItem.NextSibling().text;
    else
        pw_l6.text = "No next";
}
////////////////////////////////////////////////////////////////////////////
function click_tree()
{
    Message("Clicked on "+this.currentItem.text+" in tree");
}
////////////////////////////////////////////////////////////////////////////
function wi_onmouseover()
{
    Message("Called onMouseOver for WidgetItem "+this.text+" in tree");
}
```

Graphics (lines, circles, rectangles etc) can be drawn on Widget.LABEL and Widget.BUTTON widgets. If these methods are used the resolution of the widget is 100 units in x and y and the origin is at the top left of the widget. See the documentation below and the WidgetItem and Window classes for more details.

Constructor

new Widget(window*[Window or PopupWindow]*, type*[constant]*, left*[integer]*, right*[integer]*, top*[integer]*, bottom*[integer]*, text (optional)*[string]*)

Description

Create a new Widget object.

Arguments

- **window** (Window or PopupWindow)

Window or PopupWindow that widget will be created in

- **type** (constant)

Widget type. Can be Widget.BUTTON, Widget.CHECKBOX, Widget.COMBOBOX, Widget.LABEL, Widget.LISTBOX, Widget.RADIOBUTTON, Widget.SLIDER, Widget.TEXTBOX or Widget.TREE

- **left** (integer)

left coordinate of widget

- **right** (integer)

right coordinate of widget

- **top** (integer)

top coordinate of widget

- **bottom** (integer)

bottom coordinate of widget

- **text (optional)** (string)

Text to show on widget (optional for LABEL, BUTTON, TEXTBOX and TREE, not required for CHECKBOX, COMBOBOX, LISTBOX, RADIOBUTTON and SLIDER). For a TREE widget the text will be used as a macroTag.

Returns

Widget object

Return type

Widget

Details of functions

AddWidgetItem(item*[WidgetItem]*, position (optional)*[integer]*)

Description

Adds a WidgetItem to a ComboBox ListBox or Radiobutton Widget. Also see Widget.RemoveAllWidgetItems and Widget.RemoveWidgetItem.

Arguments

- **item** (WidgetItem)

WidgetItem to add

- **position (optional)** (integer)

Position on Widget to add the WidgetItem. Any existing WidgetItems will be shifted down as required. If omitted the WidgetItem will be added to the end of the existing ones. **Note that positions start at 0**.

Returns

No return value

Example

To add WidgetItem wi to widget w:

```
w.AddWidgetItem(wi);
```

AddWidgetItem(item*[WidgetItem]*, relationship*[constant]*, relitem*[WidgetItem]*)

Description

Adds a WidgetItem to a Tree Widget. Also see Widget.RemoveAllWidgetItems and Widget.RemoveWidgetItem.

Arguments

- **item** (WidgetItem)

WidgetItem to add

- **relationship** (constant)

What relationship (relative to relitem) to use when adding item to the Widget. Can be:
Widget.BEFORE,
Widget.AFTER or
Widget.CHILD.

- **relitem** (WidgetItem)

Existing WidgetItem to add item relative to. If relationship is Widget.CHILD then relitem can be null and then the WidgetItem will be added to the root node of the tree.

Returns

No return value

Example

To add WidgetItem wi to tree widget w after existing WidgetItem ewi:

```
w.AddWidgetItem(wi, Widget.AFTER, ewi);
```

To add WidgetItem wi to tree widget w as a child of existing WidgetItem ewi:

```
w.AddWidgetItem(wi, Widget.CHILD, ewi);
```

Circle(colour*[constant]*, fill*[boolean]*, xc*[integer]*, yc*[integer]*, radius*[integer]*)

Description

Draws a circle on the widget. Only possible for Widget.LABEL and Widget.BUTTON widgets. The coordinates are local to the Widget, not the Window. See properties xResolution and yResolution for more details. Note that the widget graphics will only be updated when the widget is redrawn. This is to allow the user to do multiple drawing commands on a widget. To force the widget to be redrawn call Show().

Arguments

- **colour** (constant)

Colour of circle. See foreground for colours.

- **fill** (boolean)

If circle should be filled or not.

- **xc** (integer)

x coordinate of centre of circle.

- **yc** (integer)

y coordinate of centre of circle.

- **radius** (integer)

radius of circle.

Returns

no return value

Example

To draw a red filled circle, radius 25, at (50, 50) on widget w:

```
w.Circle(Widget.RED, true, 50, 50, 25);
```

Clear()

Description

Clears any graphics on the widget. Only possible for Widget.LABEL and Widget.BUTTON widgets. Note that the widget graphics will only be updated when the widget is redrawn. This is to allow the user to do multiple drawing commands on a widget. To force the widget to be redrawn call Show().

Arguments

No arguments

Returns

no return value

Example

To clear any graphics for widget w:

```
w.Clear();
```

ClearSelection()

Description

Clears selection of any WidgetItems on the widget. Only possible for Widget.COMBOBOX, Widget.LISTBOX, Widget.RADIOBUTTON and Widget.TREE widgets.

Arguments

No arguments

Returns

no return value

Example

To clear selection of any WidgetItems for widget w:

```
w.ClearSelection();
```

Cross(colour (optional)*[constant]*)

Description

Draws a cross symbol on the widget. Only possible for Widget.LABEL and Widget.BUTTON widgets.

Arguments

- **colour (optional)** (constant)

Colour of cross symbol. See foreground for colours. If omitted, current foreground colour is used.

Returns

no return value

Example

To draw a red cross symbol on widget w:

```
w.Cross(Widget.RED);
```

CtrlPressed() [static]

Description

Check to see if the Ctrl key is pressed

Arguments

No arguments

Returns

true/false

Return type

Boolean

Example

To test if someone has the Ctrl key pressed:

```
if (Widget.CtrlPressed()) { ... }
```

Delete()

Description

Deletes the widget from D3PLOT (removing it from the window it is defined in) and returns any memory/resources used for the widget. This function should not normally need to be called. However, sometimes a script may want to recreate widgets in a window many times and unless the old widgets are deleted D3PLOT will reach the maximum number of widgets for a window (Options.max_widgets). To avoid this problem this method can be used to force D3PLOT to delete and return the resources for a widget. **Do not use the Widget object after calling this method**.

Arguments

No arguments

Returns

no return value

Example

To delete widget w:

```
w.Delete();
```

DirectoryIcon(line_colour*[constant]*, fill_colour*[constant]*)

Description

Draws a directory icon on the widget. Only possible for Widget.BUTTON widgets.

Arguments

- **line_colour** (constant)

Colour of lines of folder (only used in the old UI - in the new UI it will be ignored, a standard icon is always used). See foreground for colours.

- **fill_colour** (constant)

Colour of fill of folder (only used in the old UI - in the new UI it will be ignored, a standard icon is always used). See foreground for colours.

Returns

no return value

Example

To draw a directory icon on widget btn:

```
btn.DirectoryIcon(Widget.BLACK, Widget.YELLOW);
```

DumpImageString(filename*[string]*, format (optional)*[constant]*)

Description

Dumps a string representation of an image for a widget to a file in a form that can be used by Widget.ReadImageString(). Only possible for Widget.LABEL and Widget.BUTTON widgets.

Arguments

- **filename** (string)

Filename to dump string representation to

- **format (optional)** (constant)

Can be Widget.RGB8 or Widget.RGB24. Before version 15 D3PLOT only used 8 bits to store RGB (red, green and

blue) colour information for widget images. In version 15 widget images have been changed to use 24 bits to store RGB information (8 bits for red, 8 bits for green and 8 bits for blue). Both formats are supported. If omitted the new Widget.RGB24 format will be used. See Widget.ReadImageString() for more details.

Returns

no return value

Example

To dump the image data to file 'image_data' for widget w with the old 8 bit RGB representation:

```
w.DumpImageString('image_data', Widget.RGB8);
```

To dump the image data to file 'image_data' for widget w with 24 bit RGB representation:

```
w.DumpImageString('image_data', Widget.RGB24);
```

Hide()

Description

Hides the widget on the screen

Arguments

No arguments

Returns

No return value

Example

To hide widget *w*

```
w.Hide();
```

ItemAt(index*[integer]*)

Description

Returns the WidgetItem object used at *index* in this Widget. See also Widget.TotalItems() and Widget.WidgetItems(). Note that for tree Widgets the items will not be returned in the order that they are displayed in, they will be returned in the order they were added to the tree.

Arguments

- **index** (integer)

index to return WidgetItem for. **Note that indices start at 0**.

Returns

WidgetItem object.

Return type

WidgetItem

Example

To loop over the WidgetItems used in Widget *w*

```
for (i=0; i<w.TotalItems(); i++)
{
    wi = w.ItemAt(i);
}
```

Line(colour*[constant]*, x1*[integer]*, y1*[integer]*, x2*[integer]*, y2*[integer]*)

Description

Draws a line on the widget. Only possible for Widget.LABEL and Widget.BUTTON widgets. The coordinates are local to the Widget, not the Window. See properties xResolution and yResolution for more details. Note that the widget graphics will only be updated when the widget is redrawn. This is to allow the user to do multiple drawing commands on a widget. To force the widget to be redrawn call Show().

Arguments

- **colour** (constant)

Colour of line. See foreground for colours.

- **x1** (integer)

x coordinate of start of line.

- **y1** (integer)

y coordinate of start of line.

- **x2** (integer)

x coordinate of end of line.

- **y2** (integer)

y coordinate of end of line.

Returns

no return value

Example

To draw a red line from (10, 90) to (90, 10) on widget w:

```
w.Line(Widget.RED, 10, 90, 90, 10);
```

MoveWidgetItem(item*[WidgetItem]*, relationship*[constant]*, relitem*[WidgetItem or null]*)

Description

Moves an existing WidgetItem in a tree Widget. Also see Widget.RemoveAllWidgetItems and Widget.RemoveWidgetItem.

Arguments

- **item** (WidgetItem)

WidgetItem to move

- **relationship** (constant)

What relationship (relative to relitem) to use when moving item to the Widget. Can be:
Widget.BEFORE,
Widget.AFTER or
Widget.AFTER.

- **relitem** (WidgetItem or null)

Existing WidgetItem to move item relative to. If relationship is Widget.CHILD then relitem can be null and then the WidgetItem will be moved to the root node of the tree.

Returns

No return value

Example

To move WidgetItem wi in tree widget w after existing WidgetItem ewi:

```
w.MoveWidgetItem(wi, Widget.AFTER, ewi);
```

To move WidgetItem wi in tree widget w as a child of existing WidgetItem ewi:

```
w.MoveWidgetItem(wi, Widget.CHILD, ewi);
```

PixelsPerUnit() [static]

Description

Returns the number of pixels per unit coordinate. This will vary depending on the monitor D3PLOT is running on.

Arguments

No arguments

Returns

pixels/unit (real)

Return type

Number

Example

To return how many pixels there are per unit coordinate:

```
var ppu = Widget.PixelsPerUnit();
```

Polygon(colour*[constant]*, fill*[boolean]*, points*[Array of numbers]*)

Description

Draws a polygon on the widget. Only possible for Widget.LABEL and Widget.BUTTON widgets. The coordinates are local to the Widget, not the Window. See properties xResolution and yResolution for more details. Note that the widget graphics will only be updated when the widget is redrawn. This is to allow the user to do multiple drawing commands on a widget. To force the widget to be redrawn call Show(). The number of points (x, y pairs) is limited to 500. Any extra points will be ignored.

Arguments

- **colour** (constant)

Colour of polygon. See foreground for colours.

- **fill** (boolean)

If polygon should be filled or not.

- **points** (Array of numbers)

Array of point coordinates

Returns

no return value

Example

To draw a red filled triangle with corners (20, 20) and (50, 80) and (80, 20) on widget w:

```
var a = new Array(20, 20, 50, 80, 80, 20);
w.Polygon(Widget.RED, true, a);
```

Polygon(colour*[constant]*, fill*[boolean]*, x1*[integer]*, y1*[integer]*, x2*[integer]*, y2*[integer]*, ... xn*[integer]*, ... yn*[integer]*) [deprecated]

This function is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Draws a polygon on the widget. Only possible for Widget.LABEL and Widget.BUTTON widgets. The coordinates are local to the Widget, not the Window. See properties xResolution and yResolution for more details. Note that the widget graphics will only be updated when the widget is redrawn. This is to allow the user to do multiple drawing commands on a widget. To force the widget to be redrawn call Show(). The number of points (x, y pairs) is limited to 500. Any extra points will be ignored.

Arguments

- **colour** (constant)

Colour of polygon. See foreground for colours.

- **fill** (boolean)

If polygon should be filled or not.

- **x1** (integer)

x coordinate of point 1.

- **y1** (integer)

y coordinate of point 1.

- **x2** (integer)

x coordinate of point 2.

- **y2** (integer)

y coordinate of point 2.

- **... xn** (integer)

x coordinate of point n.

- **... yn** (integer)

y coordinate of point n.

Returns

no return value

Example

To draw a red filled triangle with corners (20, 20) and (50, 80) and (80, 20) on widget w:

```
w.Polygon(Widget.RED, true, 20, 20, 50, 80, 80, 20);
```

ReadImageFile(filename*[string]*, justify (optional)*[constant]*, transparent (optional)*[colour value (integer)]*, tolerance (optional)*[integer]*)

Description

Reads an image from a file to show on the widget. Only possible for Widget.LABEL and Widget.BUTTON widgets. The image will be shown on the widget underneath any text. Note that due to the way that colours are used for menus in D3PLOT only a small number of colours are available for Widget images. Black and white images will display without any issues but colour images will be displayed with a reduced set of colours.

Arguments

- **filename** (string)

Image file (BMP, GIF, JPEG or PNG) to read. To remove an image use null.

- **justify (optional)** (constant)

Widget justification. Can be a bitwise or of Widget.LEFT, Widget.RIGHT or Widget.CENTRE and Widget.TOP, Widget.MIDDLE or Widget.BOTTOM. Additionally Widget.SCALE can be used to scale the image (either reducing or enlarging it) so that it fills the widget. If omitted the default is Widget.CENTRE|Widget.MIDDLE without scaling.

- **transparent (optional)** (colour value (integer))

Transparent colour. Must be a colour returned by Colour.RGB() in D3PLOT. If given then this colour will be replaced by a transparent colour. i.e. the widget background colour will be shown. If omitted or null no transparency will be used.

- **tolerance (optional)** (integer)

Tolerance for transparent colour (0-255).
Any pixels in the image that have a red, green and blue colour value within *tolerance* of the transparent colour will be transparent.
For example if the transparent colour was given as Colour.RGB(255, 0, 0) and *tolerance* is 0 only pixels which have red value 255 **and** green value 0 **and** blue value 0 will be made transparent.
If *tolerance* is 4, pixels which have red values between 251 and 255 **and** green values between 0 and 4 **and** blue values between 0 and 4 will be made transparent.
If omitted a value of 8 will be used.

Returns

no return value

Example

To read image example.png for widget w and place it at the top left:

```
w.ReadImageFile("example.png", Widget.TOP|Widget.LEFT);
```

To read image example.png for widget w and place it at the top left, scaling it to fit the widget:

```
w.ReadImageFile("example.png", Widget.TOP|Widget.LEFT|Widget.SCALE);
```

To read image example.png for widget w and place it at the top left, replacing red with a transparent colour:

```
w.ReadImageFile("example.png", Widget.TOP|Widget.LEFT, Colour.RGB(255, 0, 0));
```

To remove an image from widget w:

```
w.ReadImageFile(null);
```

ReadImageString(string*[string]*, justify (optional)*[constant]*, transparent (optional)*[colour value (integer)]*, tolerance (optional)*[integer]*)

Description

Reads an image from a JavaScript string previously created by Widget.DumpImageString() to show on the widget. Only possible for Widget.LABEL and Widget.BUTTON widgets. The image will be shown on the widget underneath any text.
Note, prior to version 15 of D3PLOT only a small number of colours were available for Widget images. In version 14 and earlier the RGB (red, green and blue) information for each pixel in the image was packed into a single byte (8 bits) with 3 bits for red, 3 for green and 2 for blue. Widget.DumpImageString() always returned the string beginning with "RRRGGGBB_RLE" which is this 8 bit format with run length encoding. This is format Widget.RGB8.
In version 15 support for Widget images was enhanced to give 24bit support for colours. The RGB information for each pixel has 8 bits for red, 8 bits for green and 8 bits for blue. This is format Widget.RGB24.
From version 15 Widget.DumpImageString() can either return the the old 8 bit format Widget.RGB8 (string beginning with "RRRGGGBB_RLE") or return the the new 24bit format Widget.RGB24 (string beginning with "RGB24_Z"). ReadImageString supports both formats.

Arguments

- **string** (string)

String containing the image data previously created by Widget.DumpImageString(). To remove an image use null.

- **justify (optional)** (constant)

Widget justification. Can be a bitwise or of Widget.LEFT, Widget.RIGHT or Widget.CENTRE and Widget.TOP, Widget.MIDDLE or Widget.BOTTOM. Additionally Widget.SCALE can be used to scale the image (either reducing or enlarging it) so that it fills the widget. If omitted the default is Widget.CENTRE|Widget.MIDDLE without scaling.

- **transparent (optional)** (colour value (integer))

Transparent colour. Must be a colour returned by Colour.RGB() in D3PLOT. If given then this colour will be replaced by a transparent colour. i.e. the widget background colour will be shown. If omitted or null no transparency will be used.

- **tolerance (optional)** (integer)

Tolerance for transparent colour (0-255). Only used for the new 24bit format Widget.RGB24 (strings beginning with "RGB24_Z"). Ignored for the old 8 bit format Widget.RGB8 (strings beginning with "RRRGGGBB_RLE").
Any pixels in the image that have a red, green and blue colour value within *tolerance* of the transparent colour will be transparent.
For example if the transparent colour was given as Colour.RGB(255, 0, 0) and *tolerance* is 0 only pixels which have red value 255 **and** green value 0 **and** blue value 0 will be made transparent.
If *tolerance* is 4, pixels which have red values between 251 and 255 **and** green values between 0 and 4 **and** blue values between 0 and 4 will be made transparent.
If omitted a value of 8 will be used.

Returns

no return value

Example

To read image data from string s for widget w and place it at the top left:

```
w.ReadImageString(s, Widget.TOP|Widget.LEFT);
```

To read image data from string s for widget w and place it at the top left, scaling it to fit the widget:

```
w.ReadImageString(s, Widget.TOP|Widget.LEFT|Widget.SCALE);
```

To read image data from string s for widget w and place it at the top left, replacing red with a transparent colour:

```
w.ReadImageString(s, Widget.TOP|Widget.LEFT, Colour.RGB(255, 0, 0));
```

To remove an image from widget w:

```
w.ReadImageString(null);
```

Rectangle(colour*[constant]*, fill*[boolean]*, x1*[integer]*, y1*[integer]*, x2*[integer]*, y2*[integer]*)

Description

Draws a rectangle on the widget. Only possible for Widget.LABEL and Widget.BUTTON widgets. The coordinates are local to the Widget, not the Window. See properties xResolution and yResolution for more details. Note that the widget graphics will only be updated when the widget is redrawn. This is to allow the user to do multiple drawing commands on a widget. To force the widget to be redrawn call Show().

Arguments

- **colour** (constant)

Colour of rectangle. See foreground for colours.

- **fill** (boolean)

If rectangle should be filled or not.

- **x1** (integer)

x coordinate of first corner of rectangle.

- **y1** (integer)

y coordinate of first corner of rectangle.

- **x2** (integer)

x coordinate of second (opposite) corner of rectangle.

- **y2** (integer)

y coordinate of second (opposite) corner of rectangle.

Returns

no return value

Example

To draw a red filled rectangle with corners (20, 20) and (80, 80) on widget w:

```
w.Rectangle(Widget.RED, true, 20, 20, 80, 80);
```

RemoveAllWidgetItems()

Description

Removes any WidgetItems from the Widget. Also see Widget.AddWidgetItem and Widget.RemoveWidgetItem.

Arguments

No arguments

Returns

No return value

Example

To remove all WidgetItems from widget w:

```
w.RemoveAllWidgetItems();
```

RemoveWidgetItem(item*[WidgetItem]*)

Description

Removes a WidgetItem from the Widget. Also see Widget.AddWidgetItem and Widget.RemoveAllWidgetItems.

Arguments

- **item** (WidgetItem)

WidgetItem to remove

Returns

No return value

Example

To remove WidgetItem wi from widget w:

```
w.RemoveWidgetItem(wi);
```

Scroll(scroll*[constant or WidgetItem object]*)

Description

Scrolls a tree or listbox widget

Arguments

- **scroll** (constant or WidgetItem object)

How to scroll the tree/listbox widget. Can be: Widget.SCROLL_TOP, Widget.SCROLL_BOTTOM, Widget.SCROLL_UP, Widget.SCROLL_DOWN, Widget.SCROLL_PAGE_UP or Widget.SCROLL_PAGE_DOWN in which case the tree/listbox widget will be scrolled by that value or a WidgetItem, in which case the tree/listbox will be scrolled to make the WidgetItem visible, (expanding any branches as necessary to do so for the tree widget).

Returns

No return value

Example

To scroll tree widget w to the top:

```
w.Scroll(Widget.SCROLL_TOP);
```

To scroll listbox widget w so that WidgetItem wi is visible:

```
w.Scroll(wi);
```

ShiftPressed() [static]

Description

Check to see if the Shift key is pressed

Arguments

No arguments

Returns

true/false

Return type

Boolean

Example

To test if someone has the Shift key pressed:

```
if (Widget.ShiftPressed()) { ... }
```

Show()

Description

Shows the widget on the screen

Arguments

No arguments

Returns

No return value

Example

To show widget w:

```
w.Show();
```

Static()

Description

Windows have two different regions for Widgets. A 'normal' region which can be scrolled if required (if the window is made smaller scrollbars will be shown which can be used to scroll the contents) and a 'static' region at the top of the Window which is fixed and does not scroll. For an example of a static region in a Window see any of the keyword editing panels. The 'Dismiss', 'Create', 'Reset' etc buttons are in the static region. By default Widgets are put into the normal region of the Window. This method puts the Widget to the static region of the Window.

Arguments

No arguments

Returns

No return value

Example

To put widget w in the static part of the window:

```
w.Static();
```

StringLength(text*[string]*, monospace (optional)*[boolean]*, fontSize (optional)*[integer]*) [static]

Description

Returns the length of a string in Widget units. This can be used to find what size a Widget must be to be able to display the string.

Arguments

- **text** (string)

Text to find the width of

- **monospace (optional)** (boolean)

If true then width will be calculated using a monospace font. If false (default) then the normal proportional width font will be used

- **fontSize (optional)** (integer)

Calculation can be based on a defined font size, at the moment support is added only for font sizes of 6, 7, 8, 10, 12, 14, 18 and 24.

Returns

integer

Return type

Number

Example

To get the width of string 'Example':

```
var len = Widget.StringLength('Example');
```

Tick(colour (optional)*[constant]*)

Description

Draws a tick symbol on the widget. Only possible for Widget.LABEL and Widget.BUTTON widgets.

Arguments

- **colour (optional)** (constant)

Colour of tick symbol. See foreground for colours. If omitted, current foreground colour is used.

Returns

no return value

Example

To draw a red tick symbol on widget w:

```
w.Tick(Widget.RED);
```

TotalItems()

Description

Returns the number of the WidgetItem objects used in this Widget (or 0 if none used). See also Widget.ItemAt() and Widget.WidgetItems().

Arguments

No arguments

Returns

integer

Return type

Number

Example

To return the total number of WidgetItems used for Widget *w*

```
var total = w.TotalItems();
```

WidgetItems()

Description

Returns an array of the WidgetItem objects used in this Widget (or null if none used). See also Widget.ItemAt() and Widget.TotalItems().

Arguments

No arguments

Returns

Array of WidgetItem objects

Return type

Array

Example

To return WidgetItems used for Widget *w*

```
var wi = w.WidgetItems();
```

WidgetItem class

The WidgetItem class allows you to create items for combobox, listbox, radio button and tree Widgets. More...

The D3PLOT JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Member functions

- FirstChild()
- NextSibling()
- Parent()
- PreviousSibling()

WidgetItem properties

Name	Type	Description
background	constant	Widget background colour. Not used for RADIOBUTTON widgets. Can be: Widget.BLACK, Widget.WHITE, Widget.RED, Widget.GREEN, Widget.BLUE, Widget.CYAN, Widget.MAGENTA, Widget.YELLOW, Widget.DARKRED, Widget.DARKGREEN, Widget.DARKBLUE, Widget.GREY, Widget.DARKGREY, Widget.LIGHTGREY or Widget.DEFAULT
expanded	logical	If the widget item is expanded (true) or not (false) in a tree. Only available for widgetitems used in TREE widgets.
foreground	constant	Widget foreground colour. Not used for RADIOBUTTON widgets. Can be: Widget.BLACK, Widget.WHITE, Widget.RED, Widget.GREEN, Widget.BLUE, Widget.CYAN, Widget.MAGENTA, Widget.YELLOW, Widget.DARKRED, Widget.DARKGREEN, Widget.DARKBLUE, Widget.GREY, Widget.DARKGREY, Widget.LIGHTGREY or Widget.DEFAULT
hover	string	WidgetItem's hover text. Not used for RADIOBUTTON widgets.
index (read only)	integer	The index of this widgetitem in the parent widget (undefined if widgetitem is not assigned to a widget).
monospace	boolean	true if the widgetitem uses a monospace font instead of a proportional width font (default). Not used for RADIOBUTTON and TREE widgets.
onClick	function	Function to call when a widget item in a COMBOBOX, LISTBOX or TREE widget is clicked. The Widgetitem object is accessible in the function using the 'this' keyword.
onMouseOver	function	Function to call when the mouse moves over a widget item in a COMBOBOX, LISTBOX or TREE widget. The Widgetitem object is accessible in the function using the 'this' keyword.
selectable	logical	If the widget item can be selected (true) or not (false). Not used for RADIOBUTTON and TREE widgets.
selected	logical	If the widget item is selected (true) or not (false).
text	string	Widget text. If the WidgetItem is used in a tree widget then the tree will not automatically redraw (this is in case you want to change lots of tree nodes at once). In this case, use the Window Redraw method to redraw the window.
visible	logical	If the widget item is visible (true) or not (false) in a tree. A widgetitem will not be visible if it is a child of a widgetitem that is not expanded. Only available for widgetitems used in TREE widgets.

widget (read only)	Widget object	The widget that this item is defined for (null if not set)

Detailed Description

The WidgetItem class allows you to create items for combobox, listbox, radio button and tree Widgets in a Window for a graphical user interface. The following example shows how WidgetItems are used to create a Combobox Widget and how to assign callbacks to determine when the selection has been changed.

```
var items = ["D3PLOT", "PRIMER", "SHELL", "REPORTER", "T/HIS"]
// Create window
var w = new Window("Combobox example", 0.8, 1.0, 0.5, 0.6);
// A simple combobox with a few items
var cl= new Widget(w, Widget.LABEL,   1, 30, 1, 7, "Programs:");
var cb= new Widget(w, Widget.COMBOBOX,  31, 61, 1, 7);
// Add WidgetItems to Combobox
for (i=0; i<items.length; i++)
    var wi = new WidgetItem(cb, items[i]);
// A combobox with many items showing a slider.
var li= new Widget(w, Widget.LABEL,   1, 30, 8, 14, "Long list:");
var ci= new Widget(w, Widget.COMBOBOX,  31, 61, 8, 14);
// Add WidgetItems to Combobox
// As an example we also make some of the WidgetItems unselectable and
// change the background colour
for (i=1; i<=100; i++)
{
    var wi = new WidgetItem(ci, "Item "+i);
    if ( (i % 10) == 5)
    {
        wi.selectable = false;
        wi.background = Widget.WHITE;
    }
}
var e = new Widget(w, Widget.BUTTON,   1, 21, 15, 21, "Exit");
// Assign callbacks
cb.onClick  = clicked;
cb.onChange = changed;
ci.onClick  = clicked;
ci.onChange = changed;
e.onClick   = confirm_exit
// Show the window and start event loop
w.Show();
////////////////////////////////////////////////////////////////////////////
function clicked()
{
// If combobox is clicked then print the current selection
    if (this.selectedItem)
        Message("selection is currently '"+this.selectedItem.text+"'");
}
////////////////////////////////////////////////////////////////////////////
function changed()
{
// If combobox selection is changed then print the new selection
    if (this.selectedItem)
        Message("selection is now '"+this.selectedItem.text+"'");
}
////////////////////////////////////////////////////////////////////////////
function confirm_exit()
{
// Map confirm box
    var ret = Window.Question("Confirm exit", "Are you sure you want to quit?");
// If the user has answered yes then exit from the script.
    if (ret == Window.YES) Exit();
}
```

See the documentation below and the Window and Widget classes for more details.

Constructor

new WidgetItem(widget*[Widget]*, text*[string]*, selectable (optional)*[boolean]*)

Description

Create a new WidgetItem object for a combobox, listbox or radio button widget.

Arguments

- **widget** (Widget)

Combobox, listbox or radio button Widget that the widget item will be created in. This can be null in which case the WidgetItem will be created but not assigned to a Widget. It can be assigned later by using Widget.AddWidgetItem().

- **text** (string)

Text to show on widget item

- **selectable (optional)** (boolean)

If the widget item can be selected. If omitted the widget item will be selectable. Not used for RADIOBUTTON and TREE widgets.

Returns

WidgetItem object

Return type

WidgetItem

new WidgetItem(widget*[Widget]*, text*[string]*, relationship (optional)*[constant]*, relitem (optional)*[WidgetItem]*)

Description

Create a new WidgetItem object for a tree widget. If the widget argument is given and the relationship and relitem arguments are omitted then the widget item will be added as a root node in the tree. If the relationship and relitem arguments are also used then the item can be added at a specific location in the tree. Alternatively, the widget argument can be null, and the relationship and relitem arguments omitted, in which case the WidgetItem will be created but not assigned to a Widget. It can be assigned to the tree later using Widget.AddWidgetItem()

Arguments

- **widget** (Widget)

Tree Widget that the widget item will be created in or null (if the relationship and relitem arguments are omitted)

- **text** (string)

Text to show on widget item

- **relationship (optional)** (constant)

What relationship (relative to relitem) to use when adding item to the Widget. Can be:
Widget.BEFORE,
Widget.AFTER or
Widget.CHILD.

- **relitem (optional)** (WidgetItem)

Existing WidgetItem to add item relative to

Returns

WidgetItem object

Return type

WidgetItem

Details of functions

FirstChild()

Description

Returns the first child WidgetItem or null if the WidgetItem has no children. Only possible for WidgetItems used in Widget.TREE widgets.

Arguments

No arguments

Returns

WidgetItem object

Return type

WidgetItem

Example

To get the first child widget item in a tree widget for widget item wi:

```
var cwi = wi.FirstChild();
```

NextSibling()

Description

Returns the next sibling WidgetItem or null if the WidgetItem has no further siblings. Only possible for WidgetItems used in Widget.TREE widgets.

Arguments

No arguments

Returns

WidgetItem object

Return type

WidgetItem

Example

To get the next sibling widget item in a tree widget for widget item wi:

```
var pwi = wi.NextSibling();
```

Parent()

Description

Returns the parent WidgetItem or null if the WidgetItem has no parent. Only possible for WidgetItems used in Widget.TREE widgets.

Arguments

No arguments

Returns

WidgetItem object

Return type

WidgetItem

Example

To get the parent widget item in a tree widget for widget item wi:

```
var pwi = wi.Parent();
```

PreviousSibling()

Description

Returns the next sibling WidgetItem or null if the WidgetItem has no further siblings. Only possible for WidgetItems used in Widget.TREE widgets.

Arguments

No arguments

Returns

WidgetItem object

Return type

WidgetItem

Example

To get the previous sibling widget item in a tree widget for widget item wi:

```
var pwi = wi.PreviousSibling();
```

Window class

The Window class allows you to create windows for a graphical user interface. More...

The D3PLOT JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BottomBorder()
- BuildGUIFromString(guistring*[string]*)
- EndLoop()
- Error(title*[string]*, error*[string]*, buttons (optional)*[constant]*)
- GetDirectory(initial (optional)*[string]*)
- GetFile(extension (optional)*[string]*, save (optional)*[boolean]*, initial (optional)*[string]*)
- GetFilename(title*[string]*, message*[string]*, extension (optional)*[string]*, initial (optional)*[string]*, save (optional)*[boolean]*)
- GetFiles(extension (optional)*[string]*)
- GetInteger(title*[string]*, message*[string]*, initial (optional)*[integer]*)
- GetNumber(title*[string]*, message*[string]*, initial (optional)*[real]*)
- GetPassword(title*[string]*, message*[string]*)
- GetString(title*[string]*, message*[string]*, initial (optional)*[string]*)
- Information(title*[string]*, info*[string]*, buttons (optional)*[constant]*)
- MasterResolution()
- Message(title*[string]*, message*[string]*, buttons (optional)*[constant]*)
- MiddleBorder()
- Question(title*[string]*, question*[string]*, buttons (optional)*[constant]*)
- RightBorder()
- StartLoop()
- Theme(theme (optional)*[constant]*)
- TopBorder()
- UpdateGUI()
- Warning(title*[string]*, warning*[string]*, buttons (optional)*[constant]*)

Member functions

- Delete()
- Hide()
- Recompute()
- Redraw()
- Show(modal (optional)*[boolean]*)

Window constants

Name	Description
Window.CANCEL	Show CANCEL button
Window.NO	Show NO button
Window.NONMODAL	Allow Window.Error, Window.Question, Window.Warning etc windows to be non modal
Window.OK	Show OK button
Window.YES	Show YES button

Constants for Resizing/positioning

Name	Description

Window.BOTTOM	Bottom resizing/positioning of window
Window.CENTRE	Centre (horizontal) positioning of window
Window.LEFT	Left resizing/positioning of window
Window.MIDDLE	Middle (vertical) positioning of window
Window.REDUCE	Window is allowed to reduce in size when resizing
Window.RIGHT	Right resizing/positioning of window
Window.TOP	Top resizing/positioning of window

Constants for Themes

Name	Description
Window.THEME_CLASSIC	Use the Classic theme (Note: Not only the script will use this theme, the whole interface of the program will switch to classic)
Window.THEME_CURRENT	Use the current theme
Window.THEME_DARK	Use the Dark theme (Note: Not only the script will use this theme, the whole interface of the program will switch to dark)
Window.THEME_LIGHT	Use the Light theme (Note: Not only the script will use this theme, the whole interface of the program will switch to light)
Window.USE_OLD_UI_JS	Use the original, pre v17, theme (default). (Note:The interface of the program will NOT switch to old)

Window properties

Name	Type	Description
active	boolean	If true (default) then the window then the window is active and widgets in the window can be used. If false then the window is inactive and the widgets cannot be used.
background	constant	Window background colour. Can be: Widget.BLACK, Widget.WHITE, Widget.RED, Widget.GREEN, Widget.BLUE, Widget.CYAN, Widget.MAGENTA, Widget.YELLOW, Widget.DARKRED, Widget.DARKGREEN, Widget.DARKBLUE, Widget.GREY, Widget.DARKGREY, Widget.LIGHTGREY or Widget.DEFAULT
bottom	real	bottom coordinate of window in range 0.0 (bottom) to 1.0 (top)
height	real	height of window
keepOnTop	boolean	If true then the window will be kept "on top" of other windows. If false (default) then the window stacking order can be changed.
left	real	left coordinate of window in range 0.0 (left) to 1.0 (right)
maxWidgets (read only)	integer	The maximum number of widgets that can be made in this window. This can be changed **before** the window is created by using Options.max_widgets. Also see totalWidgets
onAfterShow	function	Function to call **after** a Window is shown. The Window object is accessible in the function using the 'this' keyword. This may be useful to ensure that certain actions are done after the window is shown. It can also be used to show another window so this enables multiple windows to be shown. To unset the function set the property to null.
onBeforeShow	function	Function to call **before** a Window is shown. The Window object is accessible in the function using the 'this' keyword. This may be useful to ensure that buttons are shown/hidden etc before the window is shown. Note that it cannot be used to show another window. Use onAfterShow for that. To unset the function set the property to null.

onClose	function	Function to call when a Window is closed by pressing the X on the top right of the window. This may be useful to ensure that certain actions are done when the window is closed. The Window object is accessible in the function using the 'this' keyword. If this function returns false then closing the window will be cancelled. Returning any other value (or not returning anything) will be ignored and the window will close. To unset the function set the property to null.
onHide	function	Function to call when a Window is hidden by calling Hide(). This may be useful to ensure that certain actions are done after the window is hidden. The Window object is accessible in the function using the 'this' keyword. To unset the function set the property to null.
resize	constant	Window resizing. By default when a Window is shown it is allowed to resize on all sides (left, right, top and bottom) to try to make enough room to show the Widgets. The behaviour can be changed by using this property. It can be any combination (bitwise OR) of Window.LEFT, Window.RIGHT, Window.TOP or Window.BOTTOM or 0. In addition Window.REDUCE can also be added to allow the window to reduce in size when resizing. Note that when Window.Show is called this property is set to 0 (i.e. not to resize on any side).
right	real	right coordinate of window in range 0.0 (left) to 1.0 (right)
showClose	boolean	If true (default) then a close (X) button will automatically be added on the top right of the window. If false then no close button will be shown.
shown (read only)	boolean	true if window is currently shown, false if not
title	string	Window title
top	real	top coordinate of window in range 0.0 (bottom) to 1.0 (top)
totalWidgets (read only)	integer	The total number of widgets that have been made in this window. This can be changed **before** the window is created by using Options.max_widgets. Also see maxWidgets
width	real	width of window

Detailed Description

The Window class allows you to make windows that you can place Widgets in to create a graphical user interface. The Widget class also gives a number of static methods for convenience. e.g. Window.GetInteger(). The following very simple example displays some text in a window with a button that unmaps the window when it is pressed and the user confirms that they want to exit.

```
// Create window with title "Text" from 0.8-1.0 in x and 0.5-0.6 in y
var w = new Window("Text", 0.8, 1.0, 0.5, 0.6);
// Create label widget
var l = new Widget(w, Widget.LABEL,   1, 40, 1,  7, "Press OK to exit");
// Create button widget
var e = new Widget(w, Widget.BUTTON, 11, 30, 8, 14, "OK");
// Assign the onClick callback method to the function confirm_exit'
e.onClick  = confirm_exit;
// Show the widget and start event loop
w.Show();
////////////////////////////////////////////////////////////////////////
function confirm_exit()
{
// Map confirm window
    var ret = Window.Question("Confirm exit", "Are you sure you want to quit?");
// If the user has answered Yes then exit.
    if (ret == Window.YES) Exit();
}
```

See the documentation below and the Widget class for more details.

Constructor

new Window(title*[string]*, left*[real]*, right*[real]*, bottom*[real]*, top*[real]*)

Description

Create a new Window object.

Arguments

- **title** (string)

Window title to show in title bar

- **left** (real)

left coordinate of window in range 0.0 (left) to 1.0 (right)

- **right** (real)

right coordinate of window in range 0.0 (left) to 1.0 (right)

- **bottom** (real)

bottom coordinate of window in range 0.0 (bottom) to 1.0 (top)

- **top** (real)

top coordinate of window in range 0.0 (bottom) to 1.0 (top)

Returns

Window object

Return type

Window

Example

To create a Window 'Example' in the top right half of the screen:

```
var w = new Window('Example', 0.5, 1.0, 0.5, 1.0);
```

Details of functions

BottomBorder() [static]

Description

Returns the vertical position of the bottom border (in range 0-1). This can be used to help position windows on the screen.

Arguments

No arguments

Returns

real in range 0-1

Return type

Number

Example

To obtain the position of the bottom border:

```
var b = Window.BottomBorder();
```

BuildGUIFromString(guistring*[string]*) [static]

Description

Builds a GUI from a JSON string created by the GUI builder.

Arguments

- **guistring** (string)

The string to create the GUI from

Returns

object containing the GUI

Return type

Object

Example

To create a GUI from the string json_string:

```
var gui = Window.BuildGUIFromString(json_string);
```

Delete()

Description

Deletes the window from D3PLOT and returns any memory/resources used for the window. **This function should not normally need to be called**. However, in exceptional circumstances if a script recreates windows many times D3PLOT may run out of USER objects on Microsoft Windows because of the way D3PLOT creates and shows windows. To avoid this problem this method can be used to force D3PLOT to return the resources for a window. **Do not use the Window object after calling this method**.

Arguments

No arguments

Returns

No return value

Example

To delete window w:

```
w.Delete();
```

EndLoop() [static]

Description

Ends an event loop started by Window.StartLoop()

Arguments

No arguments

Returns

No return value

Example

To end a blocking event loop started by Window.StartLoop():

```
Window.EndLoop();
```

Error(title*[string]*, error*[string]*, buttons (optional)*[constant]*) [static]

Description

Show an error message in a window.

Arguments

- **title** (string)

Title for window.

- **error** (string)

Error message to show in window. The maximum number of lines that can be shown is controlled by the Options.max_window_lines option.

- **buttons (optional)** (constant)

The buttons to use. Can be bitwise OR of Window.OK, Window.CANCEL, Window.YES or Window.NO. If this is omitted an OK button will be used. By default the window will be modal. If Window.NONMODAL is also given the window will be non-modal instead.

Returns

Button pressed

Return type

Number

Example

To show error *Critical error!\nAbort?* in window with title *Error* with Yes and No buttons:

```
var answer = Window.Error("Error", "Critical error!\nAbort?", Window.YES |
Window.NO);
if (answer == Window.YES) Exit();
```

GetDirectory(initial (optional)*[string]*) [static]

Description

Map the directory selector box native to your machine, allowing you to choose a directory. On Unix this will be a Motif selector. Windows will use the standard windows directory selector.

Arguments

- **initial (optional)** (string)

Initial directory to start from.

Returns

directory (string), (or null if cancel pressed).

Return type

String

Example

To select a directory:

```
var dir = Window.GetDirectory();
```

GetFile(extension (optional)*[string]*, save (optional)*[boolean]*, initial (optional)*[string]*) [static]

Description

Map a file selector box allowing you to choose a file. See also Window.GetFiles() and Window.GetFilename().

Arguments

- **extension (optional)** (string)

Extension to filter by.

- **save (optional)** (boolean)

If true the file selector is to be used for saving a file. If false (default) the file selector is for opening a file. Due to native operating system file selector differences, on linux new filenames can only be given when saving a file. On windows it is possible to give new filenames when opening or saving a file.

- **initial (optional)** (string)

Initial directory to start from.

Returns

filename (string), (or null if cancel pressed).

Return type

String

Example

To select a file using extension '.key':

```
var file = Window.GetFile(".key");
```

GetFilename(title*[string]*, message*[string]*, extension (optional)*[string]*, initial (optional)*[string]*, save (optional)*[boolean]*) [static]

Description

Map a window allowing you to input a filename (or select it using a file selector). OK and Cancel buttons are shown. See also Window.GetFile().

Arguments

- **title** (string)

Title for window.

- **message** (string)

Message to show in window.

- **extension (optional)** (string)

Extension to filter by.

- **initial (optional)** (string)

Initial value.

- **save (optional)** (boolean)

If true the file selector is to be used for saving a file. If false (default) the file selector is for opening a file. Due to native

operating system file selector differences, on linux new filenames can only be given when saving a file. On windows it is possible to give new filenames when opening or saving a file.

Returns

filename (string), (or null if cancel pressed).

Return type

String

Example

To create an file input window with title *Choose file* and message *Choose the file to open* and return the filename input:

```
var filename = Window.GetFilename("Choose file", "Choose the file to open");
```

GetFiles(extension (optional)*[string]*) [static]

Description

Map a file selector box allowing you to choose multiple files. See also Window.GetFile() and Window.GetFilename().

Arguments

- **extension (optional)** (string)

Extension to filter by.

Returns

Array of filenames (strings), or null if cancel pressed.

Return type

String

Example

To select multiple files using extension '.key':

```
var files = Window.GetFiles(".key");
```

GetInteger(title*[string]*, message*[string]*, initial (optional)*[integer]*) [static]

Description

Map a window allowing you to input an integer. OK and Cancel buttons are shown.

Arguments

- **title** (string)

Title for window.

- **message** (string)

Message to show in window.

- **initial (optional)** (integer)

Initial value.

Returns

value input (integer), or null if cancel pressed.

Return type

Number

Example

To create an input window with title *Input* and message *Input integer* and return the value input:

```
var value = Window.GetInteger("Input", "Input integer");
```

GetNumber(title*[string]*, message*[string]*, initial (optional)*[real]*) [static]

Description

Map a window allowing you to input a number. OK and Cancel buttons are shown.

Arguments

- **title** (string)

Title for window.

- **message** (string)

Message to show in window.

- **initial (optional)** (real)

Initial value.

Returns

value input (real), or null if cancel pressed.

Return type

Number

Example

To create an input window with title *Input* and message *Input number* and return the value input:

```
var value = Window.GetNumber("Input", "Input number");
```

GetPassword(title*[string]*, message*[string]*) [static]

Description

Map a window allowing you to input a password. OK and Cancel buttons are shown.
This is identical to Window.GetString except the string is hidden and no initial value can be given.

Arguments

- **title** (string)

Title for window.

- **message** (string)

Message to show in window.

Returns
value input (string), or null if cancel pressed.

Return type
String

Example
To create an input window with title *Input* and message *Input password* and return the value input:

```
var value = Window.GetPassword("Input", "Input password");
```

GetString(title*[string]*, message*[string]*, initial (optional)*[string]*) [static]

Description
Map a window allowing you to input a string. OK and Cancel buttons are shown.

Arguments
- **title** (string)

Title for window.

- **message** (string)

Message to show in window.

- **initial (optional)** (string)

Initial value.

Returns
value input (string), or null if cancel pressed.

Return type
String

Example
To create an input window with title *Input* and message *Input string* and return the value input:

```
var value = Window.GetString("Input", "Input string");
```

Hide()

Description
Hides (unmaps) the window. Also see the Window onHide property.

Arguments
No arguments

Returns
No return value

Example
To hide window w:

```
w.Hide();
```

Information(title*[string]*, info*[string]*, buttons (optional)*[constant]*) [static]

Description

Show information in a window.

Arguments

- **title** (string)

Title for window.

- **info** (string)

Information to show in window. The maximum number of lines that can be shown is controlled by the Options.max_window_lines option.

- **buttons (optional)** (constant)

The buttons to use. Can be bitwise OR of Window.OK, Window.CANCEL, Window.YES or Window.NO. If this is omitted an OK button will be used. By default the window will be modal. If Window.NONMODAL is also given the window will be non-modal instead.

Returns

Button pressed

Return type

Number

Example

To show information *Information* in window with title *Example* with OK and Cancel buttons:

```
var answer = Window.Information("Example", "Information", Window.OK |
Window.CANCEL);
if (answer == Window.CANCEL) Message("You pressed the Cancel button");
```

MasterResolution() [static]

Description

Returns the resolution of the master programme window in pixels

Arguments

No arguments

Returns

Array of numbers containing x and y resolution in pixels

Return type

Number

Example

To get the resolution of the main window:

```
var res = Window.MasterResolution();
```

Message(title*[string]*, message*[string]*, buttons (optional)*[constant]*) [static]

Description

Show a message in a window.

Arguments

- **title** (string)

Title for window.

- **message** (string)

Message to show in window. The maximum number of lines that can be shown is controlled by the Options.max_window_lines option.

- **buttons (optional)** (constant)

The buttons to use. Can be bitwise OR of Window.OK, Window.CANCEL, Window.YES or Window.NO. If this is omitted an OK button will be used By default the window will be modal. If Window.NONMODAL is also given the window will be non-modal instead.

Returns

Button pressed

Return type

Number

Example

To show message *Press YES or NO* in window with title *Example* with YES and NO buttons:

```
var answer = Window.Message("Example", "Press YES or NO", Window.YES |
Window.NO);
if (answer == Window.NO) Message("You pressed No");
```

MiddleBorder() [static]

Description

Returns the vertical position of the middle border (in range 0-1). The middle border is the border between the tools/keywords window and the docked windows. This can be used to help position windows on the screen.

Arguments

No arguments

Returns

real in range 0-1

Return type

Number

Example

To obtain the position of the middle border:

```
var b = Window.MiddleBorder();
```

Question(title*[string]*, question*[string]*, buttons (optional)*[constant]*) [static]

Description

Show a question in a window.

Arguments

- **title** (string)

Title for window.

- **question** (string)

Question to show in window. The maximum number of lines that can be shown is controlled by the Options.max_window_lines option.

- **buttons (optional)** (constant)

The buttons to use. Can be bitwise OR of Window.OK, Window.CANCEL, Window.YES or Window.NO. If this is omitted Yes and No button will be used. By default the window will be modal. If Window.NONMODAL is also given the window will be non-modal instead.

Returns

Button pressed

Return type

Number

Example

To show question *Do you want to continue?* in window with title *Question*:

```
var answer = Window.Question("Question", "Do you want to continue?");
if (answer == Window.NO) Message("You pressed No");
```

Recompute()

Description

Recomputes the positions of widgets in the window. If you have static widgets and 'normal' widgets in a window and you show and/or hide widgets the window needs to be recomputed to refresh the graphics, scroll bars etc. Calling this method will recompute and redraw the window.

Arguments

No arguments

Returns

No return value

Example

To recompute window w:

```
w.Recompute();
```

Redraw()

Description

Redraws the window. Sometimes if you show, hide or draw graphics on widgets the window needs to be redrawn to refresh the graphics. Calling this method will redraw the window refreshing the graphics.

Arguments

No arguments

Returns

No return value

Example

To redraw window w:

```
w.Redraw();
```

RightBorder() [static]

Description

Returns the horizontal position of the right border (in range 0-1). This can be used to help position windows on the screen.

Arguments

No arguments

Returns

real in range 0-1

Return type

Number

Example

To obtain the position of the right border:

```
var b = Window.RightBorder();
```

Show(modal (optional)*[boolean]*)

Description

Shows (maps) the window and waits for user input.

Arguments

- **modal (optional)** (boolean)

If this window is modal (true) then the user is blocked from doing anything else in D3PLOT until this window is dismissed). If non-modal (false) then the user can still use other functions in D3PLOT.
If omitted the window will be modal.
Note that making a window modal will stop interaction in all other windows and may prevent operations such as picking from working in any macros that are run from scripts.
Before version 21 D3PLOT would create a blocking event loop when the Window Show method was called (the call to the Window Show method would not return until the window was hidden again). In D3PLOT version 21 the logic has been changed so that Window Show maps the window and returns straight away. The window is managed from the main event loop in D3PLOT. In most cases this will make no differences to scripts but there may be rare cases where you specifically want the blocking behaviour. In this case use Window.StartLoop().

Returns

No return value

Example

To show window w:

```
w.Show();
```

To show window w allowing the user to use other functions in D3PLOT:

```
w.Show(false);
```

StartLoop() [static]

Description

Starts a blocking event loop in D3PLOT. Before version 21 D3PLOT would create a blocking event loop when the Window Show method was called (the call to the Window Show method would not return until the window was hidden again). In D3PLOT version 21 the logic has been changed so that Window Show maps the window and returns straight away. The window is managed from the main event loop in D3PLOT. In most cases this will make no differences to scripts but there may be rare cases where you specifically want the blocking behaviour. An example of this is using a keyout hook script, or if you want to give aprompt/question similarly to Window.Message() where you do **not** want to return to the main event loop. You want to block until the user has specified the action.
Window.StartLoop() can be used to start a local blocking, event loop. To terminate the event loop and resume execution of the script use Window.EndLoop()
Note that only a single loop can be active in D3PLOT at any one time. i.e. they cannot be nested. Window.StartLoop() should be used as sparingly as possible and only used for specific short events (e.g. something like the equivalent of Window.Message() where you really do need to block) as if it is used in one script it will prevent it from being used in another script.

Arguments

No arguments

Returns

No return value

Example

To start a blocking event loop:

```
Window.StartLoop();
```

Theme(theme (optional)*[constant]*) [static]

Description

Set or get a user interface theme.

Arguments

- **theme (optional)** (constant)

If it is provided it is used to set the current theme. Can be either Window.USE_OLD_UI_JS, Window.THEME_CURRENT, Window.THEME_DARK, Window.THEME_LIGHT, Window.THEME_CLASSIC.

Returns

Integer. When getting the theme one of: Window.USE_OLD_UI_JS, Window.THEME_DARK, Window.THEME_LIGHT, Window.THEME_CLASSIC

Return type

Number

Example

To determine the current theme:

```
var ui = Window.Theme();
      if(ui == Window.THEME_DARK)
      {
          print("Theme is dark\n");
      }
      else if(ui == Window.THEME_LIGHT)
      {
          print("Theme is light\n");
      }
      else if(ui == Window.THEME_CLASSIC)
      {
          print("Theme is classic\n");
      }
      else
      {
          print("Theme is not set\n");
      }
```

To keep the original (pre v17) appearance of your JavaScript (default):

```
Window.Theme(Window.USE_OLD_UI_JS);
```

To use the current user interface theme:

```
Window.Theme(Window.THEME_CURRENT);
```

To use the dark user interface theme:

```
Window.Theme(Window.THEME_DARK);
```

TopBorder() [static]

Description

Returns the vertical position of the top border (in range 0-1). This can be used to help position windows on the screen. This is no longer used in D3PLOT and will always be 1 but is left for backwards compatibility.

Arguments

No arguments

Returns

real in range 0-1

Return type

Number

Example

To obtain the position of the top border:

```
var b = Window.TopBorder();
```

UpdateGUI() [static]

Description

Force GUI to be updated. This function is not normally needed but if you are doing a computationally expensive operation and want to update the GUI it may be necessary as the GUI update requests are cached until there is spare time to update them. Calling this function forces any outstanding requests to be flushed.

Arguments

No arguments

Returns

No return value

Example

To force update of GUI:

```
Window.UpdateGUI();
```

Warning(title*[string]*, warning*[string]*, buttons (optional)*[constant]*) [static]

Description

Show a warning message in a window.

Arguments

- **title** (string)

Title for window.

- **warning** (string)

Warning message to show in window. The maximum number of lines that can be shown is controlled by the Options.max_window_lines option.

- **buttons (optional)** (constant)

The buttons to use. Can be bitwise OR of Window.OK, Window.CANCEL, Window.YES or Window.NO. If this is omitted an OK button will be used. By default the window will be modal. If Window.NONMODAL is also given the window will be non-modal instead.

Returns

Button pressed

Return type

Number

Example

To show warning *Title is blank\nSet to ID?* in window with title *Warning* with Yes and No buttons:

```
var answer = Window.Warning("Warning", "Title is blank\nSet to ID?", Window.YES
| Window.NO);
if (answer == Window.NO) Message("You pressed No");
```

Workflow class

The Workflow class allows you to read and write workflow files. More...

The D3PLOT JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- GetWorkflowDataFromPath(folder*[string]*, workflow_name (optional)*[string]*)
- ModelIdFromIndex(model_index*[integer]*, workflow_name (optional)*[string]*)
- ModelUnitSystemFromIndex(model_index*[integer]*, workflow_name (optional)*[string]*)
- ModelUserDataBuildFromIndex(model_index*[integer]*, workflow_name (optional)*[string]*)
- ModelUserDataFromIndex(model_index*[integer]*, workflow_name (optional)*[string]*)
- ModelUserDataProgramFromIndex(model_index*[integer]*, workflow_name (optional)*[string]*)
- ModelUserDataVersionFromIndex(model_index*[integer]*, workflow_name (optional)*[string]*)
- NumberOfSelectedModels(workflow_name (optional)*[string]*)
- Refresh()
- WorkflowDefinitionFilename(workflow_name (optional)*[string]*)
- WriteToFile(user_data*[object]*, output_filename*[string]*, workflow_definition_filename*[string]*, extra (optional)*[object]*)

Workflow constants

Constants for UnitSystem

Name	Description
Workflow.UNIT_SYSTEM_NONE	No unit system
Workflow.UNIT_SYSTEM_U1	U1 unit system (m, kg, s)
Workflow.UNIT_SYSTEM_U2	U2 unit system (mm, t, s)
Workflow.UNIT_SYSTEM_U3	U3 unit system (mm, kg, ms)
Workflow.UNIT_SYSTEM_U4	U4 unit system (mm, g, ms)
Workflow.UNIT_SYSTEM_U5	U5 unit system (ft, slug, s)
Workflow.UNIT_SYSTEM_U6	U6 unit system (m, t, s)

Detailed Description

The Workflow class gives you simple functions to read and write workflow files

Details of functions

GetWorkflowDataFromPath(folder*[string]*, workflow_name (optional)*[string]*) [static]

Description

Returns user data in the specified folder. This will first search for user data in *.json files in the specified folder and upwards in parent folders if there is no user data in the specified folder. The number of folders it will search is controlled by the preference oasys*workflow_max_upward_folder_search_depth. Additionally, if the preference oasys*workflow_user_directory_name is defined it will search in the folder with that name too in the specified folder and parent folders. Secondly, it will search for user data in keyword files in the specified folder.

Arguments

- **folder** (string)

The folder to search for user data.

- **workflow_name (optional)** (string)

The workflow name to return the user data for. If not specified it will return user data for all the workflows found.

Returns

Array of objects with the following properties:

Name	Type	Description
data	object	User data as an object
model_unit_system	integer	Model unit system associated with the user data
workflow_data_filename	string	Filename where workflow data was read from
workflow_definition	string	Workflow definition filename
workflow_name	string	Workflow name

Return type

object

ModelIdFromIndex(model_index*[integer]*, workflow_name (optional)*[string]*) [static]

Description

Returns the id of a model selected by the user by index (starting at 0).

Arguments

- **model_index** (integer)

The index of the model to return the ID for. If the workflow is run from the workflow menu and the name argument is not defined, it is the index in the list of models selected by the user. If the workflow is run from the workflow menu and the name argument is defined, it is the index of the model that has user data for the named workflow, out of the list of models selected by the user. If the workflow is run from REPORTER, it is the index in the list of all the models loaded in the session that have user data for the named workflow.

- **workflow_name (optional)** (string)

The workflow name to return the model id for. If it is not specified it uses the name of the workflow selected by the user from the workflow menu.

Returns

integer

Return type

Number

Example

To get the id of the first model selected by the user

```
var id = Workflow.ModelIdFromIndex(0);
```

To get the id of the second model and workflow name "Automotive Assessment"

```
var id = Workflow.ModelIdFromIndex(1, "Automotive Assessment");
```

ModelUnitSystemFromIndex(model_index*[integer]*, workflow_name (optional)*[string]*) [static]

Description

Returns the unit system of a model selected by the user by index (starting at 0). Will be Workflow.UNIT_SYSTEM_NONE or Workflow.UNIT_SYSTEM_U1 or Workflow.UNIT_SYSTEM_U2 or Workflow.UNIT_SYSTEM_U3 or Workflow.UNIT_SYSTEM_U4 or Workflow.UNIT_SYSTEM_U5 or Workflow.UNIT_SYSTEM_U6

Arguments

- **model_index** (integer)

The index of the model to return the unit system for. If the workflow is run from the workflow menu and the name argument is not defined, it is the index in the list of models selected by the user. If the workflow is run from the workflow menu and the name argument is defined, it is the index of the model that has user data for the named workflow, out of the list of models selected by the user. If the workflow is run from REPORTER, it is the index in the list of all the models loaded in the session that have user data for the named workflow.

- **workflow_name (optional)** (string)

The workflow name to return the unit system for.

Returns

integer

Return type

Number

Example

To get the unit system for the workflow selected from the menu, for the first model selected by the user

```
var unit_system = Workflow.ModelUnitSystemFromIndex(0);
```

To get the unit system of the second model and workflow name "Automotive Assessment"

```
var unit_system = Workflow.ModelUnitSystemFromIndex(1, "Automotive Assessment");
```

ModelUserDataBuildFromIndex(model_index*[integer]*, workflow_name (optional)*[string]*) [static]

Description

Returns the build number of the program that was used to write out the user data of a model for the selected workflow by index (starting at 0).

Arguments

- **model_index** (integer)

The index of the model to return the build number for. If the workflow is run from the workflow menu and the name argument is not defined, it is the index in the list of models selected by the user. If the workflow is run from the workflow menu and the name argument is defined, it is the index of the model that has user data for the named workflow, out of the list of models selected by the user. If the workflow is run from REPORTER, it is the index in the list of all the models loaded in the session that have user data for the named workflow.

- **workflow_name (optional)** (string)

The workflow name to return the build number for.

Returns

number

Return type

number

Example

To get the build number of the program that was used to write user data for the first model that has data for the workflow selected by the user

```
var build = Workflow.ModelUserDataBuildFromIndex(0);
```

To get the build number of the program that was used to write user data for the second model that has data for the "Automotive Assessment" workflow

```
var build = Workflow.ModelUserDataBuildFromIndex(1, "Automotive Assessment");
```

ModelUserDataFromIndex(model_index*[integer]*, workflow_name (optional)*[string]*) [static]

Description

Returns the user data associated with a model that has data for the selected workflow by index (starting at 0).

Arguments

- **model_index** (integer)

The index of the model to return the user data for. If the workflow is run from the workflow menu and the name argument is not defined, it is the index in the list of models selected by the user. If the workflow is run from the workflow menu and the name argument is defined, it is the index of the model that has user data for the named workflow, out of the list of models selected by the user. If the workflow is run from REPORTER, it is the index in the list of all the models loaded in the session that have user data for the named workflow.

- **workflow_name (optional)** (string)

The workflow name to return the user data for. If it is not specified it uses the name of the workflow selected by the user from the workflow menu.

Returns

object

Return type

Object

Example

To get the user data for the first model that has data for the workflow selected by the user

```
var user_data = Workflow.ModelUserDataFromIndex(0);
```

To get the user data for the second model that has data for the "Automotive Assessment" workflow

```
var user_data = Workflow.ModelUserDataFromIndex(1, "Automotive Assessment");
```

ModelUserDataProgramFromIndex(model_index*[integer]*, workflow_name (optional)*[string]*) [static]

Description

Returns the name of the program that was used to write out the user data of a model for the selected workflow by index (starting at 0).

Arguments

- **model_index** (integer)

The index of the model to return the program name for. If the workflow is run from the workflow menu and the name argument is not defined, it is the index in the list of models selected by the user. If the workflow is run from the workflow menu and the name argument is defined, it is the index of the model that has user data for the named workflow, out of the list of models selected by the user. If the workflow is run from REPORTER, it is the index in the list of all the models loaded in the session that have user data for the named workflow.

- **workflow_name (optional)** (string)

The workflow name to return the program name for.

Returns

string

Return type

String

Example

To get the name of the program that was used to write user data for the first model that has data for the workflow selected by the user

```
var program = Workflow.ModelUserDataProgramFromIndex(0);
```

To get the name of the program that was used to write user data for the second model that has data for the "Automotive Assessment" workflow

```
var program = Workflow.ModelUserDataProgramFromIndex(1, "Automotive
Assessment");
```

ModelUserDataVersionFromIndex(model_index*[integer]*, workflow_name (optional)*[string]*) [static]

Description

Returns the version of the program that was used to write out the user data of a model for the selected workflow by index (starting at 0).

Arguments

- **model_index** (integer)

The index of the model to return the program version for. If the workflow is run from the workflow menu and the name argument is not defined, it is the index in the list of models selected by the user. If the workflow is run from the workflow menu and the name argument is defined, it is the index of the model that has user data for the named workflow, out of the list of models selected by the user. If the workflow is run from REPORTER, it is the index in the list of all the models loaded in the session that have user data for the named workflow.

- **workflow_name (optional)** (string)

The workflow name to return the program version for.

Returns

number

Return type

number

Example

To get the version of the program that was used to write user data for the first model that has data for the workflow selected by the user

```
var version = Workflow.ModelUserDataVersionFromIndex(0);
```

To get the version of the program that was used to write user data for the second model that has data for the "Automotive Assessment" workflow

```
var version = Workflow.ModelUserDataVersionFromIndex(1, "Automotive
Assessment");
```

NumberOfSelectedModels(workflow_name (optional)*[string]*) [static]

Description

Returns the number of models selected by the user.

Arguments

- **workflow_name (optional)** (string)

The workflow name to return the number of models for. If it’s not defined the number of models that were selected by the user on the workflow menu is returned. If it’s defined and the workflow was run from the workflow menu, the number of models, out of the models selected by the user, that have data for the named workflow is returned. If it’s defined and the workflow is run from REPORTER, the number of models, out of all the models loaded in the session, that have data for the named workflow is returned.

Returns

integer

Return type

Number

Example

To get the number of models selected by the user in the workflow menu

```
var n = Workflow.NumberOfSelectedModels();
```

To get the number of models that have user data for the "Automotive Assessment" workflow

```
var n = Workflow.NumberOfSelectedModels("Automotive Assessment");
```

Refresh() [static]

Description

Scan for fresh workflow data

Arguments

No arguments

Returns

No return value

Example

```
Workflow.Refresh();
```

WorkflowDefinitionFilename(workflow_name (optional)*[string]*) [static]

Description

Returns the workflow definition filename

Arguments

- **workflow_name (optional)** (string)

The workflow name to return the workflow defintion filename for. This is required when a POST item is generated from REPORTER. If it is not specified the first workflow user data associated with the model is returned (this is the 'normal' case where a user launches a workflow from the workflow menu).

Returns

string

Return type

String

Example

To get the worklow definition filename that the script has been run with

```
var workflow_definition_filename = Workflow.WorkflowDefinitionFilename();
```

WriteToFile(user_data*[object]*, output_filename*[string]*, workflow_definition_filename*[string]*, extra (optional)*[object]*) [static]

Description

Writes a workflow to a JSON file. If the file already exists the workflow is added to the file (or updated if it is already in the file).

Arguments

- **user_data** (object)

Object containing user data required for the workflow.

- **output_filename** (string)

Filename to write to.

- **workflow_definition_filename** (string)

Filename of the workflow definition file.

- **extra (optional)** (object)

Extra workflow information

Object has the following properties:

Name	Type	Description
model_unit_system (optional)	integer	The model unit system. Can be Workflow.UNIT_SYSTEM_NONE or Workflow.UNIT_SYSTEM_U1 or Workflow.UNIT_SYSTEM_U2 or Workflow.UNIT_SYSTEM_U3 or Workflow.UNIT_SYSTEM_U4 or Workflow.UNIT_SYSTEM_U5 or Workflow.UNIT_SYSTEM_U6

Returns

No return value

Example

To write the file "C:\my_workflow.json" with some user data and the contents of the workflow definition file "C:\workflows\my_workflow_definition.json"

```
      var user_data = { part_ids: [1, 2, 10, 100], node_ids: [12, 23, 24] };
      Workflow.WriteToFile(user_data, "C:\\my_workflow.json",
"C:\\workflows\\my_workflow_definition.json");
```

XlsxWorkbook class

The XlsxWorkbook class enables writing xlsx files. More...

The D3PLOT JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Member functions

- Close()

XlsxWorkbook properties

Name	Type	Description
filename (read only)	string	Name of the xlsx file

Detailed Description

The XlsxWorkbook class provides functions to enable you to create xlsx files. The following simple example shows how to write a xlsx file:

```
 var workbook = new XlsxWorkbook("C:/temp/test.xlsx");
var worksheet = new XlsxWorksheet(workbook);
worksheet.AddText(0, 0, "Hello world!");
worksheet.AddNumber(1, 0, 1.2345);
worksheet.AddImage(2, 0, "image.png");
workbook.Close();
```

Constructor

new XlsxWorkbook(filename*[string]*)

Description

Create a new XlsxWorkbook object for writing xlsx files.

Arguments

- **filename** (string)

Filename of the xlsx file you want to write. The file will be overwritten (if it exists).

Returns

XlsxWorkbook object

Return type

XlsxWorkbook

Example

To create a new XlsxWorkbook object to write Xlsx file "/data/test/file.xlsx"

```
var workbook = new XlsxWorkbook("/data/test/file.xlsx");
```

Details of functions

Close()

Description

Close a Xlsx file

Arguments

No arguments

Returns

No return value

Example

To close Xlsx file workbook:

```
workbook.Close();
```

XlsxWorksheet class

Member functions

- AddImage(row*[integer]*, column*[integer]*, filename*[string]*, x_scale (optional)*[real]*, y_scale (optional)*[real]*)
- AddNumber(row*[integer]*, column*[integer]*, value*[number]*)
- AddText(row*[integer]*, column*[integer]*, text*[string]*)
- SetColumnProperties(column*[integer]*, width*[number]*)
- SetRowProperties(row*[integer]*, height*[number]*)

Constructor

new XlsxWorksheet(workbook*[XlsxWorkbook object]*, name (optional)*[string]*)

Description

Create a new XlsxWorksheet object for writing xlsx files.

Arguments

- **workbook** (XlsxWorkbook object)

The workbook to create the worksheet in.

- **name (optional)** (string)

The name of the worksheet. If omitted the default names 'Sheet1', 'Sheet2' etc will be used.

Returns

XlsxWorksheet object

Return type

XlsxWorksheet

Example

To create a new worksheet in workbook

```
var worksheet = new XlsxWorksheet(workbook);
```

Details of functions

AddImage(row*[integer]*, column*[integer]*, filename*[string]*, x_scale (optional)*[real]*, y_scale (optional)*[real]*)

Description

Add an image to the Xlsx file. Note that the image will not actually be read/inserted until the workbook is written by calling XlsxWorkbook.Close so you must make sure the image file exists until then.

Arguments

- **row** (integer)

The row in the xlsx file (rows start at zero)

- **column** (integer)

The column in the xlsx file (columns start at zero)

- **filename** (string)

Name of the image file you want to add to the xlsx file. The image can be in png or jpeg format.

- **x_scale (optional)** (real)

Scale factor for horizontally scaling the input image.

- **y_scale (optional)** (real)

Scale factor for vertically scaling the input image.

Returns

No return value

Example

To add image 'C:/temp/test.png' to XlsxWorksheet worksheet on the second row, third column:

```
worksheet.AddImage(1, 2, 'C:/temp/test.png');
```

AddNumber(row*[integer]*, column*[integer]*, value*[number]*)

Description

Add number to the Xlsx file

Arguments

- **row** (integer)

The row in the xlsx file (rows start at zero)

- **column** (integer)

The column in the xlsx file (columns start at zero)

- **value** (number)

Number you want to add to the xlsx file

Returns

No return value

Example

To add number 1.2345 to XlsxWorksheet worksheet on the second row, third column:

```
worksheet.AddNumber(1, 2, 1.2345);
```

AddText(row*[integer]*, column*[integer]*, text*[string]*)

Description

Add text to the Xlsx file

Arguments

- **row** (integer)

The row in the xlsx file (rows start at zero)

- **column** (integer)

The column in the xlsx file (columns start at zero)

- **text** (string)

Text you want to add to the xlsx file

Returns

No return value

Example

To add text ’test’ to XlsxWorksheet worksheet on the second row, third column:

```
worksheet.AddText(1, 2, ’test’);
```

SetColumnProperties(column*[integer]*, width*[number]*)

Description

Set the column properties in the worksheet

Arguments

- **column** (integer)

The column in the xlsx file (columns start at zero)

- **width** (number)

Width of the column to set

Returns

No return value

Example

To set the width of the third column in XlsxWorksheet worksheet to 30:

```
worksheet.SetColumnProperties(2, 30);
```

SetRowProperties(row*[integer]*, height*[number]*)

Description

Set the row properties in the worksheet

Arguments

- **row** (integer)

The row in the xlsx file (rows start at zero)

- **height** (number)

Height of the row to set

Returns

No return value

Example

To set the height of the third row in XlsxWorksheet worksheet to 20:

```
worksheet.SetRowProperties(2, 20);
```

XMLParser class

The XMLParser class enables reading data from XML files. More...

The D3PLOT JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Member functions

- Parse(filename*[string]*)

XMLParser properties

Name	Type	Description
characterDataHandler	function	Function to call when character data is found. The function will be called with 1 argument which is a string containing the character data
commentHandler	function	Function to call when a comment is found. The function will be called with 1 argument which is a string containing the text inside the comment
endCDATAHandler	function	Function to call at the end of a CDATA section. The function does not have any arguments.
endElementHandler	function	Function to call when an element end tag is found. The function will be called with 1 argument which is a string containing the name of the element
startCDATAHandler	function	Function to call at the start of a CDATA section. The function does not have any arguments.
startElementHandler	function	Function to call when an element start tag is found. The function will be called with 2 arguments. Argument 1 is a string containing the name of the element. Argument 2 is an object containing the element attributes

Detailed Description

The XMLParser class provides a stream-oriented parser to enable you to read XML files. You register callback (or handler) functions with the parser and then parse the document. As the parser recognizes parts of the document, it will call the appropriate handler for that part (if you've registered one.) The document is fed to the parser in pieces. This allows you to parse really huge documents that won't fit into memory.
There are currently 6 handlers which can be set: XMLParser.startElementHandler, XMLParser.endElementHandler, XMLParser.characterDataHandler, XMLParser.commentHandler, XMLParser.startCDATAHandler and XMLParser.endCDATAHandler.
The following simple example shows how the parser could be used.

```
// Create a new parser object
var p = new XMLParser();
// assign handlers
p.startElementHandler  = startElem;
p.endElementHandler    = endElem;
p.characterDataHandler = text;
p.commentHandler       = comment;
// parse the file
p.Parse("/data/test.xml");
////////////////////////////////////////////////////////////////////////
function startElem(name, attr)
{
// handler to be called when a start element is found
// Print element name
```

```
    Println("START: " + name);
// Print attributes
    for (n in attr)
    {
        Println(" attr: " + n + "=" + attr[n]);
    }
}
function endElem(name)
{
// handler to be called when an end element is found
// Print element name
    Println("END: " + name);
}
function text(str)
{
// handler to be called when text is found
// Print text
    Println("TEXT: '" + str + "'");
}
function comment(str)
{
// handler to be called when a comment is found
// Print comment
    Println("COMMENT: '" + str + "'");
}
```

See the documentation below for more details.

Constructor

new XMLParser()

Description

Create a new XMLParser object for reading XML files.

Arguments

No arguments

Returns

XMLParser object

Return type

XMLParser

Example

To create a new XMLParser object

```
var p = new XMLParser();
```

Details of functions

Parse(filename*[string]*)

Description

starts parsing an XML file

Arguments

- **filename** (string)

XML file to parse

Returns

No return value

Example

To parse XML file "/data/test.xml"

```
var p = new XMLParser();
p.Parse("/data/test.xml");
```

Composites

Functions and constants relating to Composites

Functions

- GetElemsInPly(ply_id*[integer]*, state_id (optional)*[integer]*)
- GetPlyIntPoint(type_code*[integer]*, item*[integer]*, ply_id*[integer]*, state_id (optional)*[integer]*)
- GetPlysInLayup(layup_id*[integer]*, state_id (optional)*[integer]*)

Details of functions

GetElemsInPly(ply_id*[integer]*, state_id (optional)*[integer]*) [static]

Description

Returns an object containing the number of elements in ply <ply_id>, the element type code, and also an array <list[ ]> of their internal indices. If there are no elements in the ply then false is returned. Ply data is only available if a .ztf file containing composite information has been read.

Arguments

- **ply_id** (integer)

The ply in which to return the list of elements. If +ve, the internal ply number starting at 1. If -ve, the external ply label. Internal numbers will be many times faster to process.

- **state_id (optional)** (integer)

State number to be used instead of the current state

Returns

Object with the following properties:

Name	Type	Description
list	array of integers	Internal element ids
nn	integer	Number of elements in list
type	integer	Element type code

Return type

object

Example

```
// Get a list of elements in ply 5
if(a = GetElemsInPly(5))
{
    var nelems = a.nn;
    for(var i=0; i<nelems; i++)
    {
        Message("Element: " + GetLabel(a.type, a.list[i]))
    }
}
```

GetPlyIntPoint(type_code*[integer]*, item*[integer]*, ply_id*[integer]*, state_id (optional)*[integer]*) [static]

Description

Return the integration point of <type/item> in ply <ply_id>. If the >type/item< is not in the ply then false is returned. Ply data is only available if a .ztf file containing composite information has been read.

Arguments

- **type_code** (integer)

A valid element type code (Currently only SHELL is valid)

- **item** (integer)

If +ve, the internal item number starting at 1. If -ve, the external label of the item. Internal numbers will be many times faster to process.

- **ply_id** (integer)

If +ve, the internal ply index. If -ve, the external ply label. Internal numbers will be many times faster to process.

- **state_id (optional)** (integer)

State number to be used instead of the current state

Returns

integer

Return type

Number

Example

```
// Find the integration point in shell #1 occupied by internal ply index 14
var ip = GetPlyIntPoint(SHELL, 1, 14));
```

GetPlysInLayup(layup_id*[integer]*, state_id (optional)*[integer]*) [static]

Description

Returns an object containing the number of plys in layup <layup_id> and an array <list[ ]> of their internal indices. If there are no plys in the layup then false is returned. Ply data is only available if a .ztf file containing composite information has been read.

Arguments

- **layup_id** (integer)

The layup in which to return the list of plys. If +ve an internal layup index, if -ve an external layup label

- **state_id (optional)** (integer)

State number to be used instead of the current state

Returns

Object with the following properties:

Name	Type	Description
list	array of integers	Array of internal ply indices
nn	integer	Number of plys in list

Return type

object

Example

```
// Print the ply labels in layup 1
if(a = GetPlysInLayup(1))
{
    var nplys = a.nn;
    for(var i=0; i<nplys; i++)
    {
        Message("Ply: " + GetLabel(CPLY, a.list[i]))
    }
}
```

Contacts

Functions and constants relating to Contacts

Functions

- GetSegmsInSurface(surface_id*[integer]*)
- SpoolNodesInSurface(surface_id*[integer]*, index*[integer]*, side*[integer]*)

Contacts constants

Name	Description
MASTER	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Master side on contact surface. Use Contact.SURFB instead [deprecated]
SLAVE	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Slave side on contact surface. Use Contact.SURFA instead [deprecated]

Details of functions

GetSegmsInSurface(surface_id*[integer]*) [static]

Description

Returns the start and end indices of the tracked and reference segments in a contact surface

Arguments

- **surface_id** (integer)

The contact surface in which to return the list of segments. If +ve, the internal surface number starting at 1. If -ve, the external surface label. Internal numbers will be many times faster to process.

Returns

Object with the following properties:

Name	Type	Description
ms_end	integer	End index of reference surface segments
ms_start	integer	Start index of reference surface segments
ss_end	integer	End index of tracked surface segments
ss_start	integer	Start index of tracked surface segments

Return type

object

Example

```
// Get the segment indices of the first contact surface in the current model
if(a = GetSegmsInSurface(1))
{
    for(var i=a.ss_start; i<=a.ss_end; i++)
    {
        Message("Tracked segments: " + GetLabel(SEGM, i));
    }
}
```

SpoolNodesInSurface(surface_id*[integer]*, index*[integer]*, side*[integer]*) [static]

Description

Spools through the nodes on a contact surface

Arguments

- **surface_id** (integer)

The contact surface in which to spool. If +ve, the internal surface number starting at 1. If -ve, the external surface label. Internal numbers will be many times faster to process.

- **index** (integer)

Index of node to get in contact surface. To setup the spool, this has to be set to zero initially

- **side** (integer)

The side of the contact surface: Contact.SURFA or Contact.SURFB

Returns

integer. If the <index> is zero then this will be the number of nodes on the surface. If it's greater than zero then the return value is the index'th node in the surface. If the call fails then false is returned.

Return type

Number

Example

```
// Get the nodes in the SURFA side of the first contact surface in the current
model
if(n = SpoolNodesInSurface(1, 0, Contact.SURFA)) //Setup spool, index=0
{
    // Spool through nodes
    for(var i=1; i<=n; i++)
    {
        nid = SpoolNodesInSurface(1, i, Contact.SURFA);
        Message("Node: " + GetLabel(NODE, nid));
    }
}
```

CutSectionOld

Functions and constants relating to CutSectionOld

Functions

- GetCutCoords(options*[object]*)
- GetCutForces(options*[object]*)
- GetCutSection(options*[object]*)
- SetCutSection(options*[object]*)

CutSectionOld constants

Constants for Action

Name	Description
NORMAL	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Positive or negative action Normal. Use Constant.NORMAL instead [deprecated]
OMIT	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Positive or negative action Omit. Use Constant.OMIT instead [deprecated]
OUTLINE	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Positive or negative action Outline (wire drawing). Use Constant.OUTLINE instead [deprecated]
TRANSPARENT	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Positive or negative action Transparent. Use Constant.TRANSPARENT instead [deprecated]

Constants for Combination

Name	Description
INTERSECTION	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Intersection mode for multiple cut directions. Use Constant.INTERSECTION instead [deprecated]
UNION	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Union mode for multiple cut directions. Use Constant.UNION instead [deprecated]

Constants for Definition

Name	Description
CONST_X	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Constant X definition method. Use Constant.CONST_X instead [deprecated]
CONST_Y	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Constant Y definition method. Use Constant.CONST_Y instead [deprecated]
CONST_Z	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Constant Z definition method. Use Constant.CONST_Z instead [deprecated]
LS_DYNA	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Ansys LS-DYNA definition method. Use Constant.LS_DYNA instead [deprecated]

N3	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. 3 nodes definition method. Use Constant.N3 instead [deprecated]
OR_AND_V	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Origin and vectors definition method. Use Constant.OR_AND_V instead [deprecated]

Constants for Space

Name	Description
BASIC	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Basic (eulerian) space. Use Constant.BASIC instead [deprecated]
DEFORMED	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Deformed (lagrangian) space. Use Constant.DEFORMED instead [deprecated]

Details of functions

GetCutCoords(options*[object]*) [static]

Description

Returns the coordinates where the cut-section cuts through element <type/item>.

Arguments

- **options** (object)

Object has the following properties:

Name	Type	Description
direction_id (optional)	integer	Index starting at 1 of the plane direction for multiple non-parallel planes. This should be at most the number of non-parallel planes (at most 3 if the maximum number of planes is defined). If omitted, the data for the first plane direction will be used.
item	integer	If +ve, the internal item number starting at 1. If -ve, the external label of the item.
model_id (optional)	integer	A valid model id that exists in <window_id>.
state_id (optional)	integer	State number to be used instead of the current state
type_code	integer	A valid type code (SHELL etc.))
window_id (optional)	integer	A valid window id.

Returns

Object with the following properties:

Name	Type	Description
n	integer	Number of places the cut-section cuts the item
x	array of reals	X coordinates where item is cut
y	array of reals	Y coordinates where item is cut
z	array of reals	Z coordinates where item is cut

Return type

object

Example

```
// Get the coordinates where the second cut plane in window 1 cuts the first
shell in model 3 in state 10.
if(a = GetCutCoords({ type_code:SHELL, item:1, window_id:1, model_id:3,
direction_id:2, state_id:10 }))
{
    var n = a.n;
    for(var i=0; i<n; i++)
    {
        Message("Coords: " + a.x[i] + ", " + a.y[i] + ", a.z[i]);
    }
}
```

GetCutForces(options*[object]*) [static]

Description

Returns the forces, moments, centroid and area of the cut section in <window_id>. See documentation for CutSection.GetForces for more detail.

Arguments

- **options** (object)

Object has the following properties:

Name	Type	Description
direction_id (optional)	integer	Index starting at 1 of the plane direction for multiple non-parallel planes. This should be at most the number of non-parallel planes (at most 3 if the maximum number of planes is defined). If omitted, the data for the first plane direction will be used.
include_ blanked (optional)	integer	0 to omit blanked elements (default), 1 to include them
model_id (optional)	integer	Model id that exists in <window_id>. If omitted the first model in the window will be used.
part_id (optional)	integer	0 all part ids considered (default), otherwise only forces in the specified part are computed. If +ve this is the internal part index, if -ve this is the external part label.
state_id (optional)	integer	State number to be used. If omitted the state of the window's current frame will be used.
window_id (optional)	integer	A valid window id. If omitted, cut section data for window 1 will be used.

Returns

Object with the following properties:

Name	Type	Description
area	real	Cut section area
centroid	array of reals	Cut section centroid, [Cx, Cy, Cz]
force	array of reals	3 Forces, [Fx, Fy, Fz]
moment	array of reals	3 Moments, [Mxx, Myy, Mzz]

Return type

object

Example

```
// Retrieve the results of the cut section in window 1 using all default
attributes.
if(a = GetCutForces({window_id: 1}))
{
    var fx = a.force[X];
    var myy = a.moment[Y];
    var area = a.area;
}
// Retrieve the results from unblanked elements in the third part using the
section
// in window 2 for state 12 in model 1 for the second non-parallel plane
direction.
var b = GetCutForces({window_id: 2, include_blanked: 0, part_id: 3, state_id:
12, model_id: 1, direction_id: 2});
```

GetCutSection(options*[object]*) [static]

Description

Retrieves all attributes of the cut section in <window_id>.

Arguments

- **options** (object)

Retrieves all attributes of the cut section in <window_id>.

Object has the following properties:

Name	Type	Description
direction_id (optional)	integer	Index starting at 1 of the plane direction for multiple non-parallel planes. This should be at most the number of non-parallel planes (at most 3 if the maximum number of planes is defined). If omitted, the data for the first plane direction will be used.
model_id (optional)	integer	A valid model id that exists in <window_id>. If omitted the state of the window's current frame will be used. This only matters if the section uses the method CONST_X, CONST_Y or CONST_Z defined by a node or N3 and it has been set to follow nodes with <follow_n>. In which case the section origin and/or vectors may change as the node(s) move.
state_id (optional)	integer	State number to be used. If omitted the state of the window's current frame will be used. This only matters if the section uses the method CONST_X, CONST_Y or CONST_Z defined by a node or N3 and it has been set to follow nodes with <follow_n>. In which case the section origin and/or vectors may change as the node(s) move.
window_id (optional)	integer	A valid window id. If omitted, cut section data for window 1 will be returned.

Returns

Object with the following properties:

Name	Type	Description
combination	integer	How positive and negative actions for multiple directions are combined. Can be UNION or INTERSECTION. Note that this only has an effect with at least 2 cut plane directions.
definition	integer	One of OR_AND_V, CONST_X, CONST_Y, CONST_Z, N3, LS_DYNA or zero if no section has been defined yet

follow_n	integer	ON if the section follows any defining node(s) as appropriate, OFF if it does not. This value is only meaningful for definition == CONST_X/Y/Z where a node in nodes[0] was supplied, and N3
multi_ custom_cut	logical	whether or not custom spacing for multiple parallel cuts is on
multi_cut	logical	whether or not multiple parallel cuts are on
negative_ action	integer	Negative action for the cut direction. Can be OMIT, OUTLINE, NORMAL or TRANSPARENT.
nneg	integer	number of negative parallel planes when multi_cut is true and multi_custom_cut is false, undefined otherwise
nodes	array of integers	For definition = CONST_X/Y/Z: nodes[0] = index of node if supplied For definition = N3: nodes[0 to 2] = indices of three N3 nodes This array will always be present and have three entries, with unused entries being set to zero.
npos	integer	number of positive parallel planes when multi_cut is true and multi_custom_cut is false, undefined otherwise
origin	array of reals	Origin coordinates
positive_ action	integer	Positive action for the cut direction. Can be OMIT, OUTLINE, NORMAL or TRANSPARENT.
space	integer	Either BASIC or DEFORMED
spacing	real or array of reals	when multi_custom_cut is false and multi_cut is true: spacing value when multi_custom_cut is true and multi_cut is true: array of offsets from the local cut origin when multi_cut is false: 0.0
status	integer	Either OFF or ON
thick_cut	logical	whether or not thick cut is on
thickness	real	thickness for thick cuts when thick_cut is true, 0.0 otherwise
visible	integer	ON if the cut direction is active, OFF otherwise.
x_axis	array of reals	Local X axis vector (normalised)
y_axis	array of reals	Local Y axis vector (normalised)
z_axis	array of reals	Local Z axis vector (normalised)

Return type

object

Example

```
// Retrieve the attributes of the cut section in window 1 at the current state
and model.
var a = GetCutSection({window_id: 1});
// Retrieve the attributes of the cut section in window 2, at state #10 in model
#3.
var a = GetCutSection({window_id: 2, state_id: 10, model_id : 3});
// Retrieve the attributes of the cut section in window 1 for the second of
several
// non-parallel plane directions.
var a = GetCutSection({window_id: 1, direction_id: 2});
```

SetCutSection(options*[object]*) [static]

Description

Sets properties of the cut section in <window_id>

Each D3PLOT window can have cut planes in up to three directions, which are by default not active. Their location, orientation and type can be defined here, and it can be turned on or off. Forces and moments from the cut sections can be obtained from function GetCutForces().

Cut section definitions are a "per window" attribute that apply to all models in the window. Thefore if the window has multiple models, and nodes are used to define the section (CONST_X, CONST_Y, CONST_Z with a node or N3), the origin and/or vectors of the section may vary for each model in the window. In addition if the coordinates of these nodes are "followed" (see the <follow_n> property on the options), then the section locations may change from state to state.

Arguments

- **options** (object)

The cut section properties which should be changed.

> All coordinates and vectors must be defined in model space, and will always form an orthogonal right handed coordinate system in which local Z is normal to the cut plane.
>
> Vector length is irrelevant (but should be well-conditioned), and the Y axis is obtained automatically from the vector cross product Z_AXIS x X_AXIS. If the Z and X axes as supplied are not at right angles the X will be updated to make it orthogonal to Y and Z.
>
> <nodes[0]> or <origin> will define the cut section origin coordinate. The array <nodes> for N3 or origin/x_axis/xy_plane will define the cut section orientation.
>
> Care must be taken when defining nodes for windows that contain multiple models. Since a node index (+ve) may resolve to a different node in each model it is usually best to use external labels (-ve) in this context to avoid ambiguity. (The speed of the external => internal lookup will not matter as this function is unlikely to be called many times.)
>
> FOLLOW_N(odes) will only have an effect if the cut section is defined by node(s), i.e. CONST_X, CONST_Y or CONST_Z with nodes[0] or N3.

Object has the following properties:

Name	Type	Description
combination (optional)	integer	How positive and negative actions for multiple directions are combined. Can be UNION or INTERSECTION. Note that this only has an effect with at least 2 cut plane directions.
definition (optional)	integer	Definition method for the cut plane direction. Can be one of OR_AND_V, CONST_X, CONST_Y, CONST_Z, N3, LS_DYNA. This can be omitted when only properties independent from the plane position are changed, for example <follow_n> or <status>.
direction_id (optional)	integer	Index starting at 1 of the plane direction for multiple non-parallel planes. Can be 1, 2 or 3. It can be either an existing direction index to update or a free index to add a new plane direction. If <direction> is omitted, the data for the first plane direction will be set by this function.

follow_n (optional)	integer	OFF or ON (default OFF). Determines whether or not a DEFORMED space section will track the motion of the node(s) used to define it. This is only meaningful for sections defined by: • N3. The motion of all three nodes will update the origin and axes at each state. • CONST_X, CONST_Y or CONST_Z where a node has been used to define the origin. The node motion will update the section origin at each state, but its axes will remain constant.
negative_ action (optional)	integer	Negative action for the cut direction. Can be OMIT, OUTLINE, NORMAL or TRANSPARENT.
nneg (optional)	integer	number of negative parallel planes for uniform spacing (required when <spacing> is a single positive number for uniform spacing)
nodes (optional)	array of integers	Array of +ve indices or -ve labels of the nodes. For CONST_X, CONST_Y or CONST_Z this should have length 1 and contain the node whose X, Y or Z coordinate defines the plane position. For N3 it should have length 3 containing the three nodes where the plane passes through. (required for <definition> CONST_X, CONST_Y or CONST_Z when origin is not defined or for <definition> N3)
normal_head (optional)	array of reals	Normal head coordinate (required for <definition> LS_DYNA)
npos (optional)	integer	number of positive parallel planes for uniform spacing (required when <spacing> is a single positive number for uniform spacing)
origin (optional)	array of reals	Coordinate array defining the origin of the plane. (for <definition> CONST_X, CONST_Y or CONST_Z when nodes is not defined or for <definition> OR_AND_V or LS_DYNA)
positive_ action (optional)	integer	Positive action for the cut direction. Can be OMIT, OUTLINE, NORMAL or TRANSPARENT.
space (optional)	integer	BASIC or DEFORMED (default). Determines whether the section is Lagrangian (BASIC) or Eulerian (DEFORMED). In the BASIC case the section is tied to the undeformed geometry and will move and distort as the model deforms, in the DEFORMED case the section is fixed in model space and the structure passes through it. For compatibility with LS_DYNA the way forces are calculated also varies with section space - see the documentation on GetCutForces().
spacing (optional)	real or array of reals	Spacing for multiple parallel cuts. If this is a single positive number, multiple parallel cuts with uniform spacing will be turned on, where <npos> and <nneg> will be required to define the number of planes in the positive and negative directions. If this is an array of real numbers, multiple parallel cuts with custom spacing will be turned on, where this array is the list of offsets from the local cut origin. If spacing is 0.0, multiple parallel cuts will be turned off. If spacing is undefined, then multiple parallel cuts will keep their existing spacing.
status (optional)	integer	OFF or ON (default OFF). Determines whether or not the cutting switch is on in the current window. The cutting switch does not need to be on in order to compute cut forces and moments.
thickness (optional)	real	Thickness for thick cuts. If positive, thick cuts will be turned on. If 0.0, thick cuts will be turned off. If thickness is undefined, then thick cuts and thickness will remain on or off as before.
visible (optional)	integer	OFF or ON (default ON). Determines whether or not the cut direction at <direction_id> is active. For it to be visible, in addition to this <visible> property also the cutting switch needs to be turned on with <status>.
window_id (optional)	integer	A valid window id or ALL for all active windows. If omitted, the cut section will be set for ALL windows.
x_axis (optional)	array of reals	X axis vector (required for <definition> OR_AND_V)
x_axis_head (optional)	array of reals	X axis head coordinate (required for <definition> LS_DYNA)
xy_plane (optional)	array of reals	XY plane vector (required for <definition> OR_AND_V)

Returns

boolean

Return type

Boolean

Example

```
// Set the cut section to "deformed" (lagrangian) space in window 1.
SetCutSection({window_id: 1, space: DEFORMED});
// Make the section in window 2 constant in Z, with its origin at
// (1,2,3). Implicitly this means a plane of constant Z = 3.
if(!SetCutSection({window_id: 2, definition: CONST_Z, origin: [1,2,3]}))
{
    ... deal with error
}
// Make a cut-section in window 3 using the LS_DYNA method with the
// origin at (0,0,0), Z axis pointing down the global X vector (head at
// (10,0,0)), and X axis down the global Y vector (0,10,0)
if(!SetCutSection({window_id: 3, definition: LS_DYNA, origin: [0,0,0], normal_
head: [10,0,0], x_axis_head: [0,10,0]}))
{
    ... deal with error
}
// In window 1 define the second direction with origin (10, 0, 0), local
// X axis (1,-1,0) and local Y axis (1,1,1). This can be called whether
// direction 2 is defined already or not.
if(!SetCutSection({window_id: 1, definition: OR_AND_V, origin: [10,0,0], x_axis:
[1,-1,0], xy_plane: [1,1,1], direction_id: 2}))
{
    ... deal with error
}
// If there is only one cut plane in window 1, turn on a thick cut with
// thickness 4.0:
if(!SetCutSection({window_id: 1, thickness: 4.0}))
{
    ... deal with error
}
// Turn off a thick cut for the second direction in window 1:
if(!SetCutSection({window_id: 1, direction_id: 2, thickness: 0.0}))
{
    ... deal with error
}
// For a single cut plane in window 1 turn on 10 parallel cuts in
// positive and 5 parallel cuts in negative direction with a uniform
// spacing of 20.0 length units:
if(!SetCutSection({window_id: 1, spacing: 20.0, npos: 10, nneg: 5}))
{
    ... deal with error
}
// For the third plane direction in window 1 turn on parallel planes
// with custom spacing such that the offsets from the local cut origin
// are -20.0, 0.0, 10.0 and 20.0 length units:
if(!SetCutSection({window_id: 1, direction_id: 3, spacing: [-20.0, 0.0, 10.0,
20.0]}))
{
    ... deal with error
}
// Turn off parallel cuts for the first direction in window 1:
if(!SetCutSection({window_id: 1, spacing: 0.0}))
{
    ... deal with error
}
// For the first plane direction use normal drawing on the positive
// side and omit items on the negative side:
if(!SetCutSection({direction_id:1, positive_action:NORMAL, negative_
action:OMIT}))
{
    ... deal with error
}
// For multiple cut directions use the intersection mode:
if(!SetCutSection({combination:INTERSECTION}))
{
    ... deal with error
}
```

Data

Functions and constants relating to Data

Functions

- GetConditionParts(component*[integer]*, value*[real]*, mode*[integer]*, int_pt (optional)*[object | integer]*, extra (optional)*[integer]*) [deprecated]
- GetContourLimit(mode*[integer]*, component (optional)*[string]*)
- GetData(component*[integer]*, type_code*[integer]*, item*[integer]*, int_pt (optional)*[object | integer]*, extra (optional)*[integer]*, fr_of_ref (optional)*[integer]*, state_id (optional)*[integer]*, dda (optional)*[integer]*, consider_blanking (optional)*[integer]*, mag_or_cur (optional)*[integer]*)
- GetMultipleData(component*[integer]*, type_code*[integer]*, item_1*[integer]*, item_2*[integer]*, int_pt (optional)*[object | integer]*, extra (optional)*[integer]*, fr_of_ref (optional)*[integer]*, state_id (optional)*[integer]*, dda (optional)*[integer]*, consider_blanking (optional)*[integer]*, mag_or_cur (optional)*[integer]*)
- GetNumOnPlanIntPts(type_code*[integer]*, item*[integer]*, state_id (optional)*[integer]*)
- GetNumberOf(type_code*[integer]*, options (optional)*[object]*)
- QueryDataPresent(component*[integer]*, type_code (optional)*[integer]*)

Data constants

Constants for Frame of Reference

Name	Description
CYLINDRICAL	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Cylindrical coordinate system. Use Constant.CYLINDRICAL instead [deprecated]
GLOBAL	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Global coordinate system. Use Constant.GLOBAL instead [deprecated]
LOCAL	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Element local coordinate system. Use Constant.LOCAL instead [deprecated]
MATERIAL	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Material axes coordinate system. Use Constant.MATERIAL instead [deprecated]
USER_ DEFINED	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. User-defined coordinate system. Use Constant.USER_DEFINED instead [deprecated]

Constants for GetNumberOf

Name	Description
CUT_ SECTION	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Number of non-parallel cut plane directions. Use Constant.CUT_SECTION instead [deprecated]
FAMILY	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Number of families. Use Constant.FAMILY instead [deprecated]
GROUP	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Number of groups. Use Type.GROUP instead [deprecated]
INCLUDE	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Number of includes. Use Constant.INCLUDE instead [deprecated]

MODEL	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Number of models. Use Type.MODEL instead [deprecated]
NEIPH	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Number of "Extra" Solid variables. Use Constant.NEIPH instead [deprecated]
NEIPS	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Number of "Extra" Shell variables. Use Constant.NEIPS instead [deprecated]
NEIPT	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Number of "Extra" Thick Shell variables. Use Constant.NEIPT instead [deprecated]
NIP_B	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Number of Beam integration points. Use Constant.NIP_B instead [deprecated]
NIP_H	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Number of Solid integration points. Use Constant.NIP_H instead [deprecated]
NIP_S	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Number of Shell integration points. Use Constant.NIP_S instead [deprecated]
NIP_T	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Number of Thick Shell integration points. Use Constant.NIP_T instead [deprecated]
N_ON_ PLAN	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Number of on-plan integration points written. Use Constant.N_ON_PLAN instead [deprecated]
N_UBMS	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Number of user-defined beam scalar components. Use Constant.N_UBMS instead [deprecated]
N_UBMV	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Number of user-defined beam vector components. Use Constant.N_UBMV instead [deprecated]
N_UNOS	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Number of user-defined node scalar components. Use Constant.N_UNOS instead [deprecated]
N_UNOV	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Number of user-defined node vector components. Use Constant.N_UNOV instead [deprecated]
N_USSS	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Number of user-defined solid/shell scalar components. Use Constant.N_USSS instead [deprecated]
N_USST	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Number of user-defined solid/shell tensor components. Use Constant.N_USST instead [deprecated]
STATE	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Number of states. Use Constant.STATE instead [deprecated]
USER	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Number of user-defined components. Use Constant.USER instead [deprecated]

Constants for Phase Angle Results

Name	Description
CURRENT_VAL	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Current value result. Use Constant.CURRENT_VAL instead [deprecated]
MAGNITUDE	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Magnitude result. Use Constant.MAGNITUDE instead [deprecated]

Details of functions

GetConditionParts(component*[integer]*, value*[real]*, mode*[integer]*, int_pt (optional)*[object | integer]*, extra (optional)*[integer]*) [static] [deprecated]

This function is deprecated in version 23.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Returns an object with all parts in current model filtered into two lists (pass_list and fail_list). Passing parts must have elements that pass the specified condition (indicated by <value> and <mode>) for the specified <component>.

For spotweld data components passing parts must have a beam or solid (hex) spotweld passing the specified condition such that the element of that weld is in the given part.

NOTE: This function only works for scalar components. If <component> is a vector or tensor component, this function will return null.

As this function is deprecated, it is recommended to use Part.GetConditionParts instead.

Arguments

- **component** (integer)

A valid component code (e.g. Component.DX, Component.SXY)

- **value** (real)

A value to compare element data against.

- **mode** (integer)

Determines if condition is Constant.LT (<), Constant.LTEQ (<=), Constant.GT (>) or Constant.GTEQ (>=) than <value>.

- **int_pt (optional)** (object) | integer

This may be one of three types:
1. A +ve integer that is an integration point id
2. One of the types Constant.TOP, Constant.MIDDLE or Constant.BOTTOM for shell surfaces only
3. An object defining both through-thickness and on-plan integration points for fully integrated shells.

Integration points are only meaningful for some element type / data component combinations:
- Shells and Tshells: Stress and strain tensors; Plastic strain; "Extra" data (if written)
- Solids: All data components if NINTSLD on the database extent binary card is 8
- Beams: 3 Stresses and 2 strains in non-resultant beam types if BEAMIP on the database extent binary card is > 0

This has become a complex data field, please see the separate section on Defining the Integration point argument below.

Where the integration point is not relevant this argument may be omitted.

If <int_pt> is not defined, this function will loop through all through thickness integration points for each element to check if the condition is fufilled.

Use zero to define a null "padding" argument

Object has the following properties:

Name	Type	Description
ip	integer	Through thickness integration point
op (optional)	integer	On plan integration point. Defaults to the first one.

- **extra (optional)** (integer)

- The "Extra" solid or shell component id for components Component.SOX or Component.SHX
- The ALE mulit-material group id for components Component.AMMG and Component.AMMS
- The sub-number for user-defined components Component.UNOS, Component.UNOV, Component.USSS, Component.USST, Component.UBMS, Component.UBMV

If any of the above component codes are used, the "extra" argument must be set to a non zero value.

Returns

Object with the following properties:

Name	Type	Description
fail_list	array of integers	List of failing parts (indices starting at 1)
failed	integer	Number of failing parts
pass_list	array of integers	List of passing parts (indices starting at 1)
passed	integer	Number of passing parts

Return type

object

Example

```
// Does the filtering for the (scalar) X stress of part elements at integration
point 2,
// checking if they are greater than 0.1
var a = GetConditionParts(Component.SXX, 0.1, Constant.GT, 2);
// Does the filtering for the yield utilisation factor of part elements (looping
through all
// integration points), checking if they are greater than or equals to 1.0
var b = GetConditionParts(Component.YUTF, 1.0, Constant.GTEQ);
```

GetContourLimit(mode*[integer]*, component (optional)*[string]*) [static]

Description

Returns the maximum/ minimum contour plot value of <component> specified for the current window. Returns null if <component> specified is not active.

Arguments

- **mode** (integer)

Constant.MIN for minimum contour plot value, or Constant.MAX for maximum contour plot value.

- **component (optional)** (string)

Contour plot component: "SCALAR_1" for Scalar 1, "SCALAR_2" for Scalar 2, "VECTOR_1" or "VECTOR" for Vector, "VECTOR_2" or "VEL" for "Vel". If only 1 component is active, <component> is optional and min/max will be evaluated for the active component. If more than 1 component is active and <component> is not specified, min/max will be evaluated for Scalar 1.

Returns

real (or null if <component> specified is not active)

Return type

Number

Example

```
// Gets the minimum contour plot value for Scalar 1
s1min = GetContourLimit(Constant.MIN, "SCALAR_1");
// Gets the maximum contour plot value for Scalar 2
s2max = GetContourLimit(Constant.MAX, "SCALAR_2");
// Gets the minimum contour plot value for Vector
v1min = GetContourLimit(Constant.MIN, "VECTOR_1");
// Gets the maximum contour plot value for "Vel"
v2max = GetContourLimit(Constant.MAX, "VECTOR_2");
```

GetData(component*[integer]*, type_code*[integer]*, item*[integer]*, int_pt (optional)*[object | integer]*, extra (optional)*[integer]*, fr_of_ref (optional)*[integer]*, state_id (optional)*[integer]*, dda (optional)*[integer]*, consider_blanking (optional)*[integer]*, mag_or_cur (optional)*[integer]*) [static]

Description

Returns the data for <component> of type <type_code> for the single <item>

WARNING: If the function arguments are grammatically correct but the requested data component is not present in the database, then 1, 3 or 6 zeros are returned as required, and no warning message is output. Therefore it is good practice to use function QueryDataPresent() to check that an optional data component is actually present in a database before attempting to extract its values.

NOTE: to return the same data for a range of items it will be much faster to call the GetMultipleData() variant of this function, described below.
In other words instead of something like this, calling GetData()for each item individually:

```
for(item=item_1; item<=item_2; item++)
{
    result = GetData(component, type, item, ...);
}
```

You can write the following to extract data into an array of results using a single call to GetMultipleData():

```
result = GetMultipleData(component, type, item_1, item_2, ...);
```

This reduces the time taken to extract data by a factor nearly equal to #items, and for a large model this can give a dramatic speed increase.

Arguments

- **component** (integer)

A valid component code (e.g. Component.DX, Component.SXY)

- **type_code** (integer)

A valid element type code (e.g. Type.SOLID, Type.SHELL)

- **item** (integer)

If +ve, the internal item number starting at 1. If -ve, the external label of the item. Internal numbers will be many times faster to process.

- **int_pt (optional)** (object) | integer

This may be one of three types:
1. A +ve integer that is an integration point id
2. One of the types Constant.TOP, Constant.MIDDLE or Constant.BOTTOM for shell surfaces only
3. An object defining both through-thickness and on-plan integration points for fully integrated shells.

Integration points are only meaningful for some element type / data component combinations:

- Shells and Tshells: Stress and strain tensors; Plastic strain; "Extra" data (if written)
- Solids: All data components if NINTSLD on the database extent binary card is 8
- Beams: 3 Stresses and 2 strains in non-resultant beam types if BEAMIP on the database extent binary card is > 0

This has become a complex data field, please see the separate section on Defining the Integration point argument below.

Where the integration point is not relevant this argument may be omitted. Use zero to define a null "padding" argument.

If the integration point is not defined it will use the integration point defined on the current GUI "data" panel, which defaults to the middle surface for shells, thick shells, and solids, and Mag All for beams, but may vary if changed by an interactive user. If consistent output from a script is required, independent of any prior interactive activity, an explicit integration point or surface should be defined.

Object has the following properties:

Name	Type	Description
ip	integer	Through thickness integration point
op (optional)	integer	On plan integration point. Defaults to the first one.

- **extra (optional)** (integer)

- The "Extra" solid or shell component id for components Component.SOX or Component.SHX
- The ALE mulit-material group id for components Component.AMMG and Component.AMMS
- The sub-number for user-defined components Component.UNOS, Component.UNOV, Component.USSS, Component.USST, Component.UBMS, Component.UBMV

If any of the above component codes are used, the "extra" argument must be set to a non zero value.

Use zero to define a null "padding" argument if this is not required

- **fr_of_ref (optional)** (integer)

The frame of reference to return values in Constant.GLOBAL, Constant.LOCAL, Constant.CYLINDRICAL, Constant.USER_DEFINED, Constant.MATERIAL. This is only necessary for directional components (eg X stress) and then only when something other than the default Constant.GLOBAL coordinate system is to be used. If omitted, or set to zero, it defaults to Constant.GLOBAL for directional components and is ignore for all others.

- **state_id (optional)** (integer)

State number to be used instead of the current state

- **dda (optional)** (integer)

Direct Disk Access flag. Either Constant.OFF (default) for normal data cacheing or Constant.ON to enable direct disk reading of data.

If turned on this reads data not currently in core memory directly from disk without loading the complete data vector for the state into core.

This should be used if you want to extract results for a few items over a range of states, since it will potentially be faster.

- **consider_blanking (optional)** (integer)

Consider blanking flag. Either Constant.OFF (default) to ignore blanking or Constant.ON to consider blanking.

This argument is relevant for nodal contact force results. By default the sum of all forces at a given node for all surfaces using that node will be returned. By blanking all but the contact surface(s) of interest and setting this argument to ON the results can be restricted to the contact surface(s) you want.

- **mag_or_cur (optional)** (integer)

Magnitude or Current Value flag. This argument is relevant for analyses with phase angle results.

Set it to Constant.MAGNITUDE to output the magnitude

Set it to Constant.CURRENT_VAL to output the current value [Magniude * cos(phase + phi)]. This is dependent on the current phi angle displayed in the graphics window and can be set using SetWindowFrame(). See example below.

If omitted, or set to zero, it defaults to Constant.MAGNITUDE.

Returns

real|Array of reals

Return type

Number

Example

```
// Returns the (scalar) X stress of internal shell #27 at integration point 2,
in the element local coordinate system.
var a = GetData(Component.SXX, Type.SHELL, 27, 2, 0, Constant.LOCAL);
// Returns the (scalar) X stress of the shell with external label 123 with the
same options.
var a = GetData(Component.SXX, Type.SHELL, -123, 2, 0, Constant.LOCAL);
// Returns an array[6] of the strain tensor in solid element#93, implicitly in
the global coordinate system.
var b = GetData(Component.ETEN, Type.SOLID, 93);
var sxx = b[Constant.XX];
var sxy = b[Constant.XY];
// Returns an array[3] of the 2nd user-defined Nodal Vector component at
internal node #inode at state #3.
var c = GetData(Component.UNOV, Type.NODE, inode,0, 2, 0, 3);
var vx = c[Constant.X];
var vy = c[Constant.Y];
var vz = c[Constant.Z];
// For an anlysis with phase angles returns the DZ displacement of internal node
#1 at the second frame of state 3.
// (Note that the state has to be set with both SetCurrentState() and a
DialogueInput() command to get CURRENT_VAL
// to work as this works off the current settings in the graphics window and
SetCurrentState() does not update the
// graphics window, it is only used internally by the Javascript interface).
SetCurrentState(3);
DialogueInput("/STATE 3");
SetWindowFrame(1, 2);
var a = GetData(Componnent.DZ, Type.NODE, 1, 0, 0, 0, 0, Constant.OFF,
Constant.OFF, Constant.CURRENT_VAL);
```

GetMultipleData(component*[integer]*, type_code*[integer]*, item_1*[integer]*, item_2*[integer]*, int_pt (optional)*[object | integer]*, extra (optional)*[integer]*, fr_of_ref (optional)*[integer]*, state_id (optional)*[integer]*, dda (optional)*[integer]*, consider_blanking (optional)*[integer]*, mag_or_cur (optional)*[integer]*) [static]

Description

Returns the data for <component> of type <type_code> for the range of items <item1 .. item2>

WARNING #1: If the function arguments are grammatically correct but the requested data component is not present in the database, then 1, 3 or 6 zeros are returned as required, and no warning message is output. Therefore it is good practice to use function QueryDataPresent() to check that an optional data component is actually present in a database before attempting to extract its values.

WARNING #2: It is possible to extract vary large quantities of data using a single call of this function. Bear in mind that JavaScript representations of values are quite bloated, for example all "numbers"are 64 bit (8 byte) floating double format, and the language imposes further overheads because of the way it organises data. For large models it may be necessary to extract large blocks of data in several smaller chunks, rather than one big one.

WARNING #3: The data return value from this function is an array of length #rows, and the subscripts of this array start at row 0. In other words the result for item_1 in the call below will be returned in results array row data[0]. When extracting results for all items of a type, for example all shells in a model, item_1 will typically be 1, and it is easy to make the mistake of expecting this to be in results array row data[1]. In addition when you extract data for vector or tensor data the result will be a two-dimensional array, aligned data[#cols][#rows]. See the examples at the bottom of this description for more information about using two-dimensional arrays.

Arguments

- **component** (integer)

A valid component code (e.g. Component.DX, Component.SXY)

- **type_code** (integer)

A valid element type code (e.g. Type.SOLID, Type.SHELL)

- **item_1** (integer)

If +ve, the internal item number starting at 1. If -ve, the external label of the item. Internal numbers will be many times faster to process.

- **item_2** (integer)

If +ve, the internal item number starting at 1. If -ve, the external label of the item. Must have the same sign as item_1, so both must be +ve or -ve. It is legal for it to be the same as item_1, in which case only values for a single item will be extracted.

- **int_pt (optional)** (object) | integer

This may be one of three types:
1. A +ve integer that is an integration point id
2. One of the types Constant.TOP, Constant.MIDDLE or Constant.BOTTOM for shell surfaces only
3. An object defining both through-thickness and on-plan integration points for fully integrated shells.

Integration points are only meaningful for some element type / data component combinations:

- Shells and Tshells: Stress and strain tensors; Plastic strain; "Extra" data (if written)
- Solids: All data components if NINTSLD on the database extent binary card is 8
- Beams: 3 Stresses and 2 strains in non-resultant beam types if BEAMIP on the database extent binary card is > 0

This has become a complex data field, please see the separate section on Defining the Integration point argument below.

Where the integration point is not relevant this argument may be omitted. Use zero to define a null "padding" argument.

If the integration point is not defined it will use the integration point defined on the current GUI "data" panel, which defaults to the middle surface for shells, thick shells, and solids, and Mag All for beams, but may vary if changed by an interactive user. If consistent output from a script is required, independent of any prior interactive activity, an explicit integration point or surface should be defined.

Object has the following properties:

Name	Type	Description
ip	integer	Through thickness integration point
op (optional)	integer	On plan integration point. Defaults to the first one.

- **extra (optional)** (integer)

- The "Extra" solid or shell component id for components Component.SOX or Component.SHX
- The ALE mulit-material group id for components Component.AMMG and Component.AMMS
- The sub-number for user-defined components Component.UNOS, Component.UNOV, Component.USSS, Component.USST, Component.UBMS, Component.UBMV

If any of the above component codes are used, the "extra" argument must be set to a non zero value.

Use zero to define a null "padding" argument if this is not required

- **fr_of_ref (optional)** (integer)

The frame of reference to return values in Constant.GLOBAL, Constant.LOCAL, Constant.CYLINDRICAL, Constant.USER_DEFINED, Constant.MATERIAL. This is only necessary for directional components (eg X stress) and then only when something other than the default Constant.GLOBAL coordinate system is to be used. If omitted, or set to zero, it defaults to Constant.GLOBAL for directional components and is ignore for all others.

- **state_id (optional)** (integer)

State number to be used instead of the current state

- **dda (optional)** (integer)

Direct Disk Access flag. Either Constant.OFF (default) for normal data cacheing or Constant.ON to enable direct disk reading of data.
If turned on this reads data not currently in core memory directly from disk without loading the complete data vector for the state into core.
This should be used if you want to extract results for a few items over a range of states, since it will potentially be faster.

- **consider_blanking (optional)** (integer)

Consider blanking flag. Either Constant.OFF (default) to ignore blanking or Constant.ON to consider blanking.

This argument is relevant for nodal contact force results. By default the sum of all forces at a given node for all surfaces using that node will be returned. By blanking all but the contact surface(s) of interest and setting this argument to ON the results can be restricted to the contact surface(s) you want.

- **mag_or_cur (optional)** (integer)

Magnitude or Current Value flag. This argument is relevant for analyses with phase angle results.
Set it to Constant.MAGNITUDE to output the magnitude
Set it to Constant.CURRENT_VAL to output the current value [Magniude * cos(phase + phi)]. This is dependent on the current phi angle displayed in the graphics window and can be set using SetWindowFrame(). See example below.

If omitted, or set to zero, it defaults to MAGNITUDE.

Returns

Object with the following properties:

Name	Type	Description

data	array of reals	• data[#rows] for data components that return a scalar value, eg DX
		• data[#cols][#rows] for data components that return a vector or tensor value, eg UNOV
		Take care when dealing with the two-dimensional array of results returned by the vector and tensor component cases, as the order in which the data is stored is [column][row]. For example if you have a tensor component then in order to extract the XY shear term for index you need to write:
		`r = GetMultipleData(args...)`
		`shear_term = r.data[Constant.XY][index];`
		Also remember that the rows in this array start at index 0, thus the results for item_1 will be row index [0] in the array of results returned, and so on.

nc	integer	The number of columns of data. 1 for scalar components, 3 for vector, 6 for tensor
nr	integer	The number of rows of data, ie how many items processed in the range <item_1 .. item_2>

Return type

object

Example

```
// Returns the (scalar) X stress of internal shells #1 to #100 inclusive at
integration
// point 2, in the element local coordinate system
a = GetMultipleData(Component.SXX, Type.SHELL, 1, 100, 2, 0, Constant.LOCAL);
sxx = a.data[0];  // X stress in first shell
sxx = a.data[99]; // X stress in 100th shell
// Returns an array[6] of the strain tensor in solid elements #1 to #100,
implicitly in the
// global coordinate system.
b = GetMultipleData(Component.ETEN, Type.SOLID, 1, 100);
sxx = b.data[Constant.XX][0];  // X strain in 1st solid
sxy = b.data[Constant.XY][99]; // XY strain in 100th solid
// Returns an array[3] of the 2nd user-defined Nodal Vector component at nodes
with external
// labels 1 to 100 at state #3.
// Note that when a range of external labels is supplied, ie -ve values for
<item_1> and <item_2>,
// you should check the .nr return value to see how many rows of results were
actually returned, since
// if there are gaps in that label range the result may not be |item_2| - |item_
1| + 1.
c = GetMultipleData(Component.UNOV, Type.NODE, -1, -100, 0, 2, 0, 3);
nres = c.nr; // Number of rows of data returned
vx = c.data[Constant.X][0]; // X component for 1st node
vy = c.data[Constant.Y][1]; // Y component for 2nd node
vz = c.data[Constant.Z][2]; // Z component for 3rd node
```

GetNumOnPlanIntPts(type_code*[integer]*, item*[integer]*, state_id (optional)*[integer]*) [static]

Description

Returns the number of on plan points in an element in the current model

Arguments

- **type_code** (integer)

A type code (either Type.SHELL or Type.TSHELL)

- **item** (integer)

If +ve, the internal item number starting at 1. If -ve, the external label of the item.

- **state_id (optional)** (integer)

The state to use instead of the current state. Only necessary in adaptively remeshed analyses.

Returns

integer

Return type

Number

Example

```
// Return the number of on plan points in the first SHELL element in the current
model
var a = GetNumOfPlanIntPts(Type.SHELL, 1);
```

GetNumberOf(type_code*[integer]*, options (optional)*[object]*) [static]

Description

Returns the number of items of type_code in the current model

Note that in adaptively remeshed models the current family may affect the number of nodes and elements returned. The family of the current state will be used unless you supply the optional state_id argument, in which case the family of that state will be used.

Notes:

The number of models returned by GetNumberOf(Type.MODEL) is actually the number of active and inactive model "slots" in the database, including those currently not in use. This means that it will always return the highest model number that has been used to date.

Therefore the following sequence:

- Read in (say) three models M1 to M3
- Delete models M1 and M2, leaving only M3 in use

Will result in GetNumberOf(Type.MODEL) returning the value 3.

You can use SetCurrentModel(model_id) to attempt to set a model and examine its return value to see whether it succeeded or failed:

```
n = GetNumberOf(Type.MODEL);
for(i=1; i<=n; i++)
{
    if(SetCurrentModel(i))  // TRUE if present
    {
        do something
    }
}
```

Arguments

- **type_code** (integer)

A valid type code or a 'GetNumberOf' constant

- **options (optional)** (object)

Object has the following properties:

Name	Type	Description
state_id (optional)	integer	The state to use instead of the current state. Only necessary in adaptively remeshed analyses.

Returns

integer

Return type

Number

Example

```
// Return the number of models
var a = GetNumberOf(Type.MODEL);
// Return the number of shell integration points
var a = GetNumberOf(Constant.NIP_S);
// Return the number of solid elements in family of state 20
var a = GetNumberOf(Type.SOLID, { state_id:20 });
```

QueryDataPresent(component*[integer]*, type_code (optional)*[integer]*) [static]

Description

Returns true if data <component> is present in the current model's database, otherwise false. For some data components that are switchable the <type_code> must also be supplied, these are listed below.

Arguments

- **component** (integer)

A valid component code (e.g. Component.DX, Component.SXY)

- **type_code (optional)** (integer)

One of the type codes Type.SOLID, Type.SHELL or Type.TSHELL if the component is:

- Stress tensor derived, e.g. SXX, ... SVON
- Strain tensor derived, e.g. EXX, ... EVON
- Effective plastic strain, EPL
- Strain rate, ERATE

Returns

boolean

Return type

Boolean

Example

```
// Returns true if Effective Plastic Strain exists for solid
var a = QueryDataPresent(Component.EPL, Type.SOLID);
// Returns true if nodal temperatures exist
var a = QueryDataPresent(Component.TEMP);
```

DataComponents

Functions and constants relating to DataComponents

NOTE: LSDA-derived data are only available if both a ZTF file (which provides geometry and topology) and an LSDA file (which provides results) have been read. Also an attempt to extract a data component that does not match the element type, for example spring force for a spotweld, will return 0.0. These components and their names are configured dynamically from the "d3plot.components" file, which will be be in one or more of the $OA_ADMIN, $OA_INSTALL or $OA_HOME directories. If this file is updated further components may become available.

The D3PLOT JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

DataComponents constants

Constants for ALE Data Components

Name	Description
ADENS	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. ALE density. Use Component.ADENS instead [deprecated]
ADOMF	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. ALE dominant fraction. Use Component.ADOMF instead [deprecated]
AMMG	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. ALE multi-material group id. Use Component.AMMG instead [deprecated]
AMMS	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. ALE multi-material group mass. Use Component.AMMS instead [deprecated]

Constants for Basic and Integrated Beam Force/Moment Data Components

Name	Description
BFMV	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Force and moment vector [BFX, BFY, BFZ, BMXX, BMYY, BMZZ]. Use Component.BFMV instead [deprecated]
BFR	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Force magnitude. Use Component.BFR instead [deprecated]
BFX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. X axial force. Use Component.BFX instead [deprecated]
BFY	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Y axial force. Use Component.BFY instead [deprecated]
BFZ	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Z axial force. Use Component.BFZ instead [deprecated]

BMXX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. XX torsional moment. Use Component.BMXX instead [deprecated]
BMYY	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. YY bending moment. Use Component.BMYY instead [deprecated]
BMZZ	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. ZZ bending moment. Use Component.BMZZ instead [deprecated]
BRM	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Moment magnitude. Use Component.BRM instead [deprecated]

Constants for Basic and Integrated Beam Strain Data Components

Name	Description
BEAX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Axial strain. Use Component.BEAX instead [deprecated]
BEP	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Effective plastic strain. Use Component.BEP instead [deprecated]

Constants for Basic and Integrated Beam Stress Data Components

Name	Description
BSXX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Axial stress. Use Component.BSXX instead [deprecated]
BSYX or BSXY	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. XY shear stress. Use Component.BSXY instead [deprecated]
BSZX or BSXZ	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. XZ shear stress. Use Component.BSZX instead [deprecated]

Constants for Belytschko-Schwer Resultant Beam Energy Data Components

Name	Description
BAED	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Axial energy density. Use Component.BAED instead [deprecated]
BAEN	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Axial energy. Use Component.BAEN instead [deprecated]
BBED	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Bending energy density. Use Component.BBED instead [deprecated]
BIE	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Internal energy. Use Component.BIE instead [deprecated]
BIED	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Internal energy density. Use Component.BIED instead [deprecated]

Constants for Belytschko-Schwer Resultant Beam Moment Data

Components

Name	Description
BMY1	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Y moment at end 1. Use Component.BMY1 instead [deprecated]
BMY2	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Y moment at end 2. Use Component.BMY2 instead [deprecated]
BMZ1	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Z moment at end 1. Use Component.BMZ1 instead [deprecated]
BMZ2	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Z moment at end 2. Use Component.BMZ2 instead [deprecated]

Constants for Belytschko-Schwer Resultant Beam Rotation Data Components

Name	Description
BRXX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Torsional rotation. Use Component.BRXX instead [deprecated]
BRY1	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Y rotation at end 1. Use Component.BRY1 instead [deprecated]
BRY2	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Y rotation at end 2. Use Component.BRY2 instead [deprecated]
BRZ1	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Z rotation at end 1. Use Component.BRZ1 instead [deprecated]
BRZ2	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Z rotation at end 2. Use Component.BRZ2 instead [deprecated]

Constants for Belytschko-Schwer Resultant Beam Strain Data Components

Name	Description
BPE1	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Plastic energy at end 1. Use Component.BPE1 instead [deprecated]
BPE2	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Plastic energy at end 2. Use Component.BPE2 instead [deprecated]
BSAX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Total axial strain. Use Component.BSAX instead [deprecated]

Constants for Contact Surface Data Components (if a .ctf file has been read)

Name	Description

CAREA	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Contact segment area. Use Component.CAREA instead [deprecated]
CFGX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Contact global X force. Use Component.CFGX instead [deprecated]
CFGY	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Contact global Y force. Use Component.CFGY instead [deprecated]
CFGZ	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Contact global Z force. Use Component.CFGZ instead [deprecated]
CFLX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Contact local X force. Use Component.CFLX instead [deprecated]
CFLY	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Contact local Y force. Use Component.CFLY instead [deprecated]
CFLZ	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Contact local Z force. Use Component.CFLZ instead [deprecated]
CFM	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Contact force magnitude. Use Component.CFM instead [deprecated]
CSN	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Contact normal stress. Use Component.CSN instead [deprecated]
CST	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Contact tangential stress. Use Component.CST instead [deprecated]
CSX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Contact local X stress. Use Component.CSX instead [deprecated]
CSY	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Contact local Y stress. Use Component.CSY instead [deprecated]

Constants for Element Plastic Strain Data Components

Name	Description
EPL	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Effective plastic strain. Use Component.EPL instead [deprecated]
ERATE	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Strain rate. Use Component.ERATE instead [deprecated]
PEMAG	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Plastic strain magnitude. Use Component.PEMAG instead [deprecated]

Constants for Element Plastic Strain Derived Data Components

Name	Description

PEAV	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Average plastic strain. Use Component.PEAV instead [deprecated]
PEMAX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Max principal plastic strain. Use Component.PEMAX instead [deprecated]
PEMID	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Middle principal plastic strain. Use Component.PEMID instead [deprecated]
PEMIN	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Min principal plastic strain. Use Component.PEMIN instead [deprecated]
PEMS	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Max plastic shear strain. Use Component.PEMS instead [deprecated]

Constants for Element Plastic Strain Tensor Data Components

Name	Description
PETEN	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Plastic strain tensor [EXX, EYY, EZZ, EXY, EYZ, EZX]. Use Component.PETEN instead [deprecated]
PEXX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. X Plastic strain. Use Component.PEXX instead [deprecated]
PEXY or PEYX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. XY Plastic shear strain. Use Component.PEXY instead [deprecated]
PEYY	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Y Plastic strain. Use Component.PEYY instead [deprecated]
PEYZ or PEZY	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. XY Plastic shear strain. Use Component.PEYZ instead [deprecated]
PEZX or PEXZ	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. ZX Plastic shear strain. Use Component.PEZX instead [deprecated]
PEZZ	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Z Plastic strain. Use Component.PEZZ instead [deprecated]

Constants for Element Strain Derived Data Components

Name	Description
E2MAX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. 2D (in-plane) max principal strain. Use Component.E2MAX instead [deprecated]
E2MIN	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. 2D (in-plane) min principal strain. Use Component.E2MIN instead [deprecated]
E2SHEAR	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. 2D (in-plane) max shear strain. Use Component.E2SHEAR instead [deprecated]

EAV	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Average strain. Use Component.EAV instead [deprecated]
EMAX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Max prinicipal strain. Use Component.EMAX instead [deprecated]
EMID	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Middle prinicipal strain. Use Component.EMID instead [deprecated]
EMIN	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Min prinicipal strain. Use Component.EMIN instead [deprecated]
EMS	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Max shear strain. Use Component.EMS instead [deprecated]
ENGMAJ	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Engineering Major strain. Use Component.ENGMAJ instead [deprecated]
ENGMIN	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Engineering Minor strain. Use Component.ENGMIN instead [deprecated]
ENGTHK	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Engineering Thickness strain. Use Component.ENGTHK instead [deprecated]
ERATIO	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. 2D (in-plane) principal strain ratio. Use Component.ERATIO instead [deprecated]
EVON	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. von Mises strain. Use Component.EVON instead [deprecated]

Constants for Element Strain Tensor Data Components

Name	Description
ETEN	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Strain tensor [EXX, EYY, EZZ, EXY, EYZ, EZX]. Use Component.ETEN instead [deprecated]
EXX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. X strain. Use Component.EXX instead [deprecated]
EXY or EYX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. XY shear strain. Use Component.EXY instead [deprecated]
EYY	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Y strain. Use Component.EYY instead [deprecated]
EYZ or EZY	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. YZ shear strain. Use Component.EYZ instead [deprecated]
EZX or EXZ	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. ZX shear strain. Use Component.EZX instead [deprecated]
EZZ	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Z strain. Use Component.EZZ instead [deprecated]

Constants for Element Stress Derived Data Components

Name	Description
LODE_A	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Lode angle. Use Component.LODE_A instead [deprecated]
LODE_P	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Lode parameter. Use Component.LODE_P instead [deprecated]
LODE_PA	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Lode parameter alt. Use Component.LODE_PA instead [deprecated]
S2MAX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. 2D (in-plane) max principal stress. Use Component.S2MAX instead [deprecated]
S2MIN	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. 2D (in-plane) min principal stress. Use Component.S2MIN instead [deprecated]
S2SHEAR	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. 2D (in-plane) max shear stress. Use Component.S2SHEAR instead [deprecated]
SAV	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Average stress (pressure). Use Component.SAV instead [deprecated]
SMAX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Max principal stress. Use Component.SMAX instead [deprecated]
SMID	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Middle principal stress. Use Component.SMID instead [deprecated]
SMIN	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Min principal stress. Use Component.SMIN instead [deprecated]
SMS	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Max shear stress. Use Component.SMS instead [deprecated]
SVON	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. von Mises stress. Use Component.SVON instead [deprecated]
TRI	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Triaxiality. Use Component.TRI instead [deprecated]
YUTF	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Yield Utilisation Factor. Use Component.YUTF instead [deprecated]
YUTP	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Yield Utilisation Percentage. Use Component.YUTP instead [deprecated]

Constants for Element Stress Tensor Data Components

Name	Description

STEN	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Stress tensor [SXX, SYY, SZZ, SXY, SYZ, SZX]. Use Component.STEN instead [deprecated]
SXX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. X stress. Use Component.SXX instead [deprecated]
SXY or SYX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. XY stress. Use Component.SXY instead [deprecated]
SYY	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Y stress. Use Component.SYY instead [deprecated]
SYZ or SZY	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. YZ stress. Use Component.SYZ instead [deprecated]
SZX or SXZ	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. ZX stress. Use Component.SZX instead [deprecated]
SZZ	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Z stress. Use Component.SZZ instead [deprecated]

Constants for Element Thermal Strain Derived Data Components

Name	Description
TEAV	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Average thermal strain. Use Component.TEAV instead [deprecated]
TEMAX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Max principal thermal strain. Use Component.TEMAX instead [deprecated]
TEMID	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Middle principal thermal strain. Use Component.TEMID instead [deprecated]
TEMIN	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Min principal thermal strain. Use Component.TEMIN instead [deprecated]
TEMS	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Max thermal shear strain. Use Component.TEMS instead [deprecated]

Constants for Element Thermal Strain Tensor Data Components

Name	Description
TETEN	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Thermal strain tensor [EXX, EYY, EZZ, EXY, EYZ, EZX]. Use Component.TETEN instead [deprecated]
TEXX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. X Thermal strain. Use Component.TEXX instead [deprecated]
TEXY or PEYX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. XY Thermal shear strain. Use Component.TEXY instead [deprecated]

TEYY	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Y Thermal strain. Use Component.TEYY instead [deprecated]
TEYZ or TEZY	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. XY Thermal shear strain. Use Component.TEYZ instead [deprecated]
TEZX or TEXZ	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. ZX Thermal shear strain. Use Component.TEZX instead [deprecated]
TEZZ	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Z Thermal strain. Use Component.TEZZ instead [deprecated]

Constants for Extra Solid and Shell Data Components

Name	Description
SHX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Extra shell and thick shell data. Use Component.SHX instead [deprecated]
SOX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Extra solid data. Use Component.SOX instead [deprecated]

Constants for Global Energy Data Components

Name	Description
GIE	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Internal energy. Use Component.GIE instead [deprecated]
GKE	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Kinetic energy. Use Component.GKE instead [deprecated]
GTE	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Total energy. Use Component.GTE instead [deprecated]

Constants for Global Mass Data Components

Name	Description
GMASS	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Mass. Use Component.GMASS instead [deprecated]

Constants for Global Momentum Data Components

Name	Description
GMM	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Momentum magnitude. Use Component.GMM instead [deprecated]
GMX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. X Momentum. Use Component.GMX instead [deprecated]
GMY	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Y Momentum. Use Component.GMY instead [deprecated]

GMZ	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Z Momentum. Use Component.GMZ instead [deprecated]

Constants for Global Velocity Data Components

Name	Description
GVM	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Velocity magnitude. Use Component.GVM instead [deprecated]
GVX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. X Velocity. Use Component.GVX instead [deprecated]
GVY	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Y Velocity. Use Component.GVY instead [deprecated]
GVZ	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Z Velocity. Use Component.GVZ instead [deprecated]

Constants for LSDA (binout) Database Cross Section Data Components

Name	Description
XSEC_A	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Database X-sect area. Use Component.XSEC_A instead [deprecated]
XSEC_F	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Database X-sect force (vector data). Use Component.XSEC_F instead [deprecated]
XSEC_M	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Database X-sect moment (vector data). Use Component.XSEC_M instead [deprecated]

Constants for LSDA (binout) Retractor Data Components

Name	Description
RT_F	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Retractor force. Use Component.RT_F instead [deprecated]
RT_P	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Retractor pull-out. Use Component.RT_P instead [deprecated]

Constants for LSDA (binout) SPC Data Components

Name	Description
SPC_F	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. SPC force (vector data at nodes). Use Component.SPC_F instead [deprecated]
SPC_M	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. SPC moment (vector data at nodes). Use Component.SPC_M instead [deprecated]

Constants for LSDA (binout) Seatbelt Data Components

Name	Description
SB_F	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Seatbelt axial force. Use Component.SB_F instead [deprecated]
SB_L	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Seatbelt length. Use Component.SB_L instead [deprecated]

Constants for LSDA (binout) Slipring Data Components

Name	Description
SR_P	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Slipring pull-through. Use Component.SR_P instead [deprecated]

Constants for LSDA (binout) Spotweld Data Components

Name	Description
SW_F	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Spotweld axial force. Use Component.SW_F instead [deprecated]
SW_FAIL	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Spotweld failure. Use Component.SW_FAIL instead [deprecated]
SW_S	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Spotweld shear force. Use Component.SW_S instead [deprecated]
SW_TIME	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Spotweld failure time. Use Component.SW_TIME instead [deprecated]
SW_TRSN	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Spotweld torsion moment. Use Component.SW_TRSN instead [deprecated]

Constants for LSDA (binout) Spring Data Components

Name	Description
SP_E	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Spring elongation. Use Component.SP_E instead [deprecated]
SP_F	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Spring axial force. Use Component.SP_F instead [deprecated]
SP_M	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Spring torsional moment. Use Component.SP_M instead [deprecated]
SP_R	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Spring rotation. Use Component.SP_R instead [deprecated]

Constants for Material Data Components for PARTs and Part-based elems (needs .ZTF file)

Name	Description
DENS	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Material density. Use Component.DENS instead [deprecated]
FSTRN	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Failure strain. Use Component.FSTRN instead [deprecated]
PRAT	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Poisson's ratio. Use Component.PRAT instead [deprecated]
YMOD	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Young's modulus. Use Component.YMOD instead [deprecated]
YSTRS	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Yield stress. Use Component.YSTRS instead [deprecated]

Constants for Nastran OP2 Beam Data Components

Name	Description
BENL	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Energy loss. Use Component.BENL instead [deprecated]
BENLD	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Energy loss density. Use Component.BENLD instead [deprecated]
BENLP	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Energy loss percentage. Use Component.BENLP instead [deprecated]
BKEN	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Kinetic energy. Use Component.BKEN instead [deprecated]
BKEND	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Kinetic energy density. Use Component.BKEND instead [deprecated]
BKENP	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Kinetic energy percentage. Use Component.BKENP instead [deprecated]
BSEN	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Strain energy. Use Component.BSEN instead [deprecated]
BSEND	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Strain energy density. Use Component.BSEND instead [deprecated]
BSENP	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Strain energy percentage. Use Component.BSENP instead [deprecated]

Constants for Nodal Data Components

Name	Description
AM	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Acceleration magnitude. Use Component.AM instead [deprecated]

AV	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Acceleration vector [AX, AY, AZ]. Use Component.AV instead [deprecated]
AX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. X acceleration. Use Component.AX instead [deprecated]
AY	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Y acceleration. Use Component.AY instead [deprecated]
AZ	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Z acceleration. Use Component.AZ instead [deprecated]
BV	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Basic (undeformed) vector [BX, BY, BZ]. Use Component.BV instead [deprecated]
BX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Basic (undeformed) X coordinate. Use Component.BX instead [deprecated]
BY	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Basic (undeformed) Y coordinate. Use Component.BY instead [deprecated]
BZ	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Basic (undeformed) Z coordinate. Use Component.BZ instead [deprecated]
CV	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Current vector [CX, CY, CZ]. Use Component.CV instead [deprecated]
CX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Current X coordinate. Use Component.CX instead [deprecated]
CY	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Current Y coordinate. Use Component.CY instead [deprecated]
CZ	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Current Z coordinate. Use Component.CZ instead [deprecated]
DM	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Displacement magnitude. Use Component.DM instead [deprecated]
DV	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Displacement vector [DX, DY, DZ]. Use Component.DV instead [deprecated]
DX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. X displacement. Use Component.DX instead [deprecated]
DY	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Y displacement. Use Component.DY instead [deprecated]
DZ	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Z displacement. Use Component.DZ instead [deprecated]
RAM	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Rotation acceleration magnitude. Use Component.RAM instead [deprecated]

RAV	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Rotation acceleration vector [RAX, RAY, RAZ]. Use Component.RAV instead [deprecated]
RAX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. X rotation acceleration. Use Component.RAX instead [deprecated]
RAY	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Y rotation acceleration. Use Component.RAY instead [deprecated]
RAZ	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Z rotation acceleration. Use Component.RAZ instead [deprecated]
RDM	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Rotation displacement magnitude. Use Component.RDM instead [deprecated]
RDV	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Rotation displacement vector [RDX, RDY, RDZ]. Use Component.RDV instead [deprecated]
RDX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. X rotation displacement. Use Component.RDX instead [deprecated]
RDY	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Y rotation displacement. Use Component.RDY instead [deprecated]
RDZ	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Z rotation displacement. Use Component.RDZ instead [deprecated]
RVM	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Rotation velocity magnitude. Use Component.RVM instead [deprecated]
RVV	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Rotation velocity vector [RVX, RVY, RVZ]. Use Component.RVV instead [deprecated]
RVX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. X rotation velocity. Use Component.RVX instead [deprecated]
RVY	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Y rotation velocity. Use Component.RVY instead [deprecated]
RVZ	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Z rotation velocity. Use Component.RVZ instead [deprecated]
VM	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Velocity magnitude. Use Component.VM instead [deprecated]
VV	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Velocity vector [VX, VY, VZ]. Use Component.VV instead [deprecated]
VX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. X velocity. Use Component.VX instead [deprecated]
VY	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Y velocity. Use Component.VY instead [deprecated]

VZ	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Z velocity. Use Component.VZ instead [deprecated]

Constants for Shell and Solid Data Components

Name	Description
AREA	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Area. Use Component.AREA instead [deprecated]
DTDT	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. dTemp / dTime. Use Component.DTDT instead [deprecated]
EDEN	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Internal energy density. Use Component.EDEN instead [deprecated]
ENL	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Energy loss (Nastran OP2 results only). Use Component.ENL instead [deprecated]
ENLD	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Energy loss density (Nastran OP2 results only). Use Component.ENLD instead [deprecated]
ENLP	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Energy loss percentage (Nastran OP2 results only). Use Component.ENLP instead [deprecated]
HGEN	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Hourglass energy. Use Component.HGEN instead [deprecated]
JS_C_MASS	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Mass. Use Component.EMASS instead [deprecated]
KEN	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Kinetic energy (Nastran OP2 results only). Use Component.KEN instead [deprecated]
KEND	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Kinetic energy density (Nastran OP2 results only). Use Component.KEND instead [deprecated]
KENP	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Kinetic energy percentage (Nastran OP2 results only). Use Component.KENP instead [deprecated]
MADD	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Added mass. Use Component.MADD instead [deprecated]
RFX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. X force resultant. Use Component.RFX instead [deprecated]
RFXY	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. XY force resultant. Use Component.RFXY instead [deprecated]
RFY	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Y force resultant. Use Component.RFY instead [deprecated]
RMX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. MX moment resultant. Use Component.RMX instead [deprecated]

RMXY	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. MXY moment resultant. Use Component.RMXY instead [deprecated]
RMY	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. MY moment resultant. Use Component.RMY instead [deprecated]
RQX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. XZ shear force resultant. Use Component.RQX instead [deprecated]
RQY	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. YZ shear force resultant. Use Component.RQY instead [deprecated]
RVOL	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Relative volume (solid). Use Component.RVOL instead [deprecated]
SEN	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Strain energy (Nastran OP2 results only). Use Component.SEN instead [deprecated]
SEND	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Strain energy density (Nastran OP2 results only). Use Component.SEND instead [deprecated]
SENP	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Strain energy percentage (Nastran OP2 results only). Use Component.SENP instead [deprecated]
TBOT	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Nodal (shell) bottom surface temperature. Use Component.TBOT instead [deprecated]
TEMP	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Nodal temperature. Use Component.TEMP instead [deprecated]
TFM	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Temperature magnitude. Use Component.TFM instead [deprecated]
TFV	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Temperature vector [TFX, TFY, TFZ]. Use Component.TFV instead [deprecated]
TFX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. X temperature flux. Use Component.TFX instead [deprecated]
TFY	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Y temperature flux. Use Component.TFY instead [deprecated]
TFZ	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Z temperature flux. Use Component.TFZ instead [deprecated]
THK	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Thickness. Use Component.THK instead [deprecated]
TMID	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Nodal (shell) middle surface temperature. Use Component.TMID instead [deprecated]
TSTP	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Timestep. Use Component.TSTP instead [deprecated]

TTOP	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Nodal (shell) top surface temperature. Use Component.TTOP instead [deprecated]
VOL	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Volume (solid). Use Component.VOL instead [deprecated]

Elements

Functions and constants relating to Elements

Functions

- GetElemAxes(type_code*[integer]*, item*[integer]*, state_id (optional)*[integer]*)
- GetElemBetaAngle(type_code*[integer]*, item*[integer]*, ply_id*[integer]*, int_pnt (optional)*[integer]*, state_id (optional)*[integer]*)
- GetElemsAtNode(node*[integer]*, type_code*[integer]*, state_id (optional)*[integer]*)
- GetElemsInPart(part_id*[integer]*, state_id (optional)*[integer]*)
- GetTopology(type_code*[integer]*, item*[integer]*, state_id (optional)*[integer]*)

Details of functions

GetElemAxes(type_code*[integer]*, item*[integer]*, state_id (optional)*[integer]*) [static]

Description

Returns the local axes of the element in model space, expressed as direction cosines in a 2d array

Arguments

- **type_code** (integer)

A valid element type code (SOLID, BEAM, SHELL or TSHELL)

- **item** (integer)

If +ve, the internal item number starting at 1. If -ve, the external label of the item. Internal numbers will be many times faster to process.

- **state_id (optional)** (integer)

State number to be used instead of the current state

Returns

2d array of reals. The cosines of the element in model space as a two-dimensional array, subscripts [row][col], with cosines organised in rows.

Spelled out in detail for results array R this means:

- X axis cosines: R[X][X], R[X][Y], R[X][Z]
- Y axis cosines: R[Y][X], R[Y][Y], R[Y][Z]
- Z axis cosines: R[Z][X], R[Z][Y], R[Z][Z]

Return type

Number

Example

```
// Return the direction cosines of shell 1
var r = GetElemAxes(SHELL, 1);
var yz_cosine = r[Y][Z];
```

GetElemBetaAngle(type_code*[integer]*, item*[integer]*, ply_id*[integer]*, int_pnt (optional)*[integer]*, state_id (optional)*[integer]*) [static]

Description

Returns the beta angle (in degrees) at either the ply id or integration point number on element <item> of <type_code>

If <ply_id> is non-zero then <int_pnt> can be omitted or set to zero.

If <ply_id> is zero then <int_pnt> must be defined and non-zero.

When working with <ply_id> if the ply does not exist in the element, then false is returned.

When working with <int_pnt> a value will always be returned, but this will be zero if no beta angle is defined for the element / int_pnt combination.

Ply data is only available if a .ztf file containing composite information has been read.

Arguments

- **type_code** (integer)

A valid element type code (Currently only SHELL is valid)

- **item** (integer)

If +ve, the internal item number starting at 1. If -ve, the external label of the item. Internal numbers will be many times faster to process.

- **ply_id** (integer)

If +ve, the internal ply index. If -ve, the external ply label. Internal numbers will be many times faster to process. Set to zero if <int_pt> is to be used instead.

- **int_pnt (optional)** (integer)

Integration point in the range 1 - maxint, required if <ply_id> is zero.

- **state_id (optional)** (integer)

State number to be used instead of the current state

Returns

real

Return type

Number

Example

```
// Return the beta angle in shell 1 for ply external label 13
var beta = GetElemBetaAngle(SHELL, 1, -13);
// Return the beta angle in shell 1 at integration point 7
var beta = GetElemBetaAngle(SHELL, 1, 0, 7);
```

GetElemsAtNode(node*[integer]*, type_code*[integer]*, state_id (optional)*[integer]*) [static]

Description

Returns an object containing the number of elements of <type> at <node>, and also an array <list[ ]> of their internal indices. If there are no elements of <type> at the node then false is returned.

Arguments

- **node** (integer)

The node at which to return the list of elements. If +ve, the internal node number starting at 1. If -ve, the external node

label. Internal numbers will be many times faster to process.

- **type_code** (integer)

A valid element type code (SOLID etc.)

- **state_id (optional)** (integer)

State number to be used instead of the current state

Returns

Object with the following properties:

Name	Type	Description
list	array of integers	Internal element ids
nn	integer	Number of elements in list

Return type

object

Example

```
// Get a list of shell elements at node 5
if(a = GetElemsAtNode(5, SHELL))
{
    var nelems = a.nn;
    var e1 = a.list[0];
    var e2 = a.list[1];
}
```

GetElemsInPart(part_id*[integer]*, state_id (optional)*[integer]*) [static]

Description

Returns an object containing the number of elements in part <part_id>, the element type code, and also an array <list[ ]> of their internal indices. If there are no elements in the part then false is returned.

Arguments

- **part_id** (integer)

The part in which to return the list of elements. If +ve, the internal part number starting at 1. If -ve, the external part label. Internal numbers will be many times faster to process.

- **state_id (optional)** (integer)

State number to be used instead of the current state

Returns

Object with the following properties:

Name	Type	Description
list	array of integers	Internal element ids
nn	integer	Number of elements in list
type	integer	Element type code

Return type

object

Example

```
// Get a list of elements in part 5
if(a = GetElemsInPart(5))
{
    var nelems = a.nn;
    for(var i=0; i<nelems; i++)
    {
        Message("Element: " + GetLabel(a.type, a.list[i]))
    }
}
```

GetTopology(type_code*[integer]*, item*[integer]*, state_id (optional)*[integer]*) [static]

Description

Returns the topology list for internal <item> of type <type_code>. This should only be used for element types which have nodal topologies.

Arguments

- **type_code** (integer)

A valid element type code (SOLID etc.)

- **item** (integer)

If +ve, the internal item number starting at 1. If -ve, the external label of the item. Internal numbers will be many times faster to process.

- **state_id (optional)** (integer)

State number to be used instead of the current state

Returns

Object with the following properties:

Name	Type	Description
nn	integer	Number of nodes in topology list
pid	integer	Internal part id for part-based elements, otherwise zero
top	array of integers	Internal node ids

Return type

object

Example

```
// Get the topology of internal shell 27
var a = GetTopology(SHELL, 27);
var nnodes = a.nn;
var n1 = a.top[0];
var n2 = a.top[1];
var pid = a.pid;
```

Groups

Functions and constants relating to Groups

Functions

- GetGroupInfo(group_id*[integer]*)

Details of functions

GetGroupInfo(group_id*[integer]*) [static]

Description

Returns information about a group in the current model

Arguments

- **group_id** (integer)

Group number

Returns

Object with the following properties:

Name	Type	Description
label	integer	The label of the group
name	string	The name of the group

Return type

object

Example

```
// Print the name of the first group in the current model
var info = GetGroupInfo(1);
Print("Group name = " + info.name + "\n");
```

Includes

Functions and constants relating to Includes

Functions

- GetIncludeInfo(include_id*[integer]*)

Details of functions

GetIncludeInfo(include_id*[integer]*) [static]

Description

Returns information about an include file in the current model

Arguments

- **include_id** (integer)

Include number

Returns

Object with the following properties:

Name	Type	Description
label	integer	The label of the include file
name	string	The name of the include file
parent	integer	The parent include file (0 if main file)

Return type

object

Example

```
// Print the name of the first include file in the current model
var info = GetIncludeInfo(1);
Print("Include name = " + info.name + "\n");
```

IntegrationPoints

Detailed Description

Defining the Integration point argument in GetData() and GetMultipleData() has become a complex field. This section describes the different ways it can be defined and what values it can be set to.

Defining the Integration point argument <int_pnt> for use in GetData() and GetMultipleData()

Recent developments in Ansys LS-DYNA mean that the permutations of integration point and element type have become quite complex if some of the more detailed output options are chosen. Therefore this section has been split up by element type.

Shells and Thick shell elements:

<int_pt> may be ignored for "whole element" data components such as strain energy density, thickness and force/moment resultants; it may also be ignored for nodally derived data such as displacement, etc. For all other data components the table below should be used.

MAXINT on the *DATABASE_EXTENT_BINARY card:	
>= 0	Shell output has MAXINT (or 3 if MAXINT = 0) integration points through the thickness of the element, results are averaged at the centre on plan. Stress tensors, plastic strains and any "extra" variables are written out for each integration point. Other results: force and moment resultants, thickness, energy are written for the whole element. <int_pnt> may be: • A +ve integer to obtain results at integration point <int_pnt> • One of Constant.TOP, Constant.MIDDLE or Constant.BOTTOM to obtain results at that surface Integration point numbering starts at the bottom of the element with point 1, working upwards in the +ve local Z direction.

< 0	Shell output is written for |MAXINT| integration points through the thickness, and also for on-plan (in-plane) integration points.
	<int_pnt> needs to be an object which specifies members "ip" for the through-thickness integration point and "op" for the on-plan. For example you might write the Javascript:
	`var ip_arg = { }; // Create an empty object`
	`ip_arg.ip = 2;  // Through-thickness integration point #2`
	`ip_arg.op = 4;  // On plan integration point #4`
	Which would specify though thickness integration point 2, and on-plan integration point 4.
	If the on plan point is not specified it will default to using the first one.
	The function GetNumOnPlanIntPts() can be used to get the number of on plan points for an element.

You need to take particular care with the quantity of through-thickness integration points. Ansys LS-DYNA allows them to be defined as follows:

- On the *SECTION_SHELL card, field NIP
- On the *PART_COMPOSITE card
- On the *ELEMENT_SHELL_COMPOSITE card

Regardless of the above Ansys LS-DYNA writes results at MAXINT (or 3 if MAXINT is zero) through-thickness integration points for each element, so when extracting data for a specific integration point you may need to know something about its *ELEMENT, *SECTION and *PART definition.

If you have written a ZTF file from PRIMER then D3PLOT will "know about" these cards and will be able to determine the TOP, MIDDLE and BOTTOM surface of each element.

If you have **not** written a ZTF file then it is strongly recommended that you only use Constant.TOP, Constant.MIDDLE or Constant.BOTTOM if MAXINT = 0 or 3, since otherwise D3PLOT cannot "know" reliably which surface is which in a given element, and will treat all elements as having MAXINT points.

Solid elements:

If you are extracting nodally-derived results (eg displacement) then <int_pnt> can be ignored. Otherwise use the table below:

NINTSLD on the *DATABASE_EXTENT_BINARY card:	
Is zero or 1	Solid element output is averaged at the element centre. <int_pnt> will be ignored and may be set to zero.
Is 8	Solid element data is written at all 8 integration points. <int_pnt> should be the integration point number

Although NINTSLD = 8 case writes 8 data "slots" for a solid element integration points, degenerate types (wedges and tetrahedra) will only populate a subset of those slots.

Beam elements:

If you are extracting nodally derived results (eg displacement), or your beams are resultant Belytschko-Schwer type then <int_pnt> can be ignored. Otherwise use the table below:

BEAMIP on the *DATABASE_EXTENT_BINARY card:	
= 0	No extra beam data has been written <int_pnt> will be ignored and may be set to zero.
> 0	Extra data has been written for BEAMIP integration points <int_pnt> should be the integration point number if you are extracting an "extra" data component. Otherwise it can be ignored and may be set to zero.

"Extra" beam data is specified as follows:

- It only applies to Hughes-Liu ("integrated") beam formulations.
- It is 5 values: axial stress, 2 shear stresses, effective plastic strain, axial strain. (However see remark 2 on *Database Extent Binary in the Ansys LS-DYNA manual for some special cases)

Other beam data, the forces and moments and any extra "resultant" Belytschko-Schwer plastic results, are written for the beam as a whole and ignore <int_pnt>

Other element types:

These all ignore the <int_pnt> argument which may be set to zero or omitted.

Labels

Functions and constants relating to Labels

Functions

- GetLabel(type_code*[integer]*, item*[integer]*, state_id (optional)*[integer]*)

Details of functions

GetLabel(type_code*[integer]*, item*[integer]*, state_id (optional)*[integer]*) [static]

Description

Returns the external label of internal <item> of type <type_code>

Arguments

- **type_code** (integer)

A valid type code (e.g. NODE, SOLID, SHELL)

- **item** (integer)

The internal number starting from 1

- **state_id (optional)** (integer)

State number to use instead of the current state

Returns

integer

Return type

Number

Example

```
// Get the label of the 27th internal node
var label = GetLabel(NODE, 27);
```

Materials

Functions and constants relating to Materials

Functions

- GetMid(type_code*[integer]*, item*[integer]*, layer_id (optional)*[integer]*, state_id (optional)*[integer]*)

Details of functions

GetMid(type_code*[integer]*, item*[integer]*, layer_id (optional)*[integer]*, state_id (optional)*[integer]*) [static]

Description

Returns the external material id of internal <item> of type <type_code>

Use of this function requires that material data be present, which means that a .ztf file must have been read. If the optional <layer_id> argument is used the element must be in a part using a *PART_COMPOSITE definition.

If the material number is requested for a (composite) layer that does not exist in this item a value of zero is returned. No warning message is issued in this situation since experience has shown that this is a common occurrence and excessive warning messages are a nuisance.

Arguments

- **type_code** (integer)

PART or a valid part-based element type code (e.g. SOLID, SHELL)

- **item** (integer)

If +ve, the internal item number starting at 1. If -ve, the external label of the item. Internal numbers will be many times faster to process.

- **layer_id (optional)** (integer)

For composites the layer number 1 - n.

- **state_id (optional)** (integer)

State number to be used instead of the current state

Returns

integer

Return type

Number

Example

```
// Return the external material id of *PART 2
var a = GetMid(PART, 2);
// Return the external material id of the 3rd layer of *PART_COMPOSITE 12
var b = GetMid(PART, 12, 3);
// Return the external material id of the 27th internal shell
var c = GetMid(SHELL, 27);
// Return the external material id of the 2nd layer of internal shell 100
// Assumes the part is a *PART_COMPOSITE
var d = GetMid(SHELL, 100, 2);
```

Models

Functions and constants relating to Models

Functions

- GetModelInfo(model_id (optional)*[integer]*, family_id (optional)*[integer]*)
- ModelExists(model_id*[integer]*)
- SetCurrentModel(model_id*[integer]*)

Details of functions

GetModelInfo(model_id (optional)*[integer]*, family_id (optional)*[integer]*) [static]

Description

Returns information about filenames in the current model, or in model_id if specified. It is an error to define model_id that is not currently in use.

Notes

The vast majority of analyses do not use adaptive remeshing and the family_id argument can be ignored. When it is given:

Family id 0 is the base analysis

Family id 1 is the first remesh, ie name_aa

... and so on

Arguments

- **model_id (optional)** (integer)

Model number. The current model is used if unspecified or zero

- **family_id (optional)** (integer)

Family number (starting from zero). The family number of an adaptive remesh analysis

Returns

Object with the following properties:

Name	Type	Description
ctf_name	string	the full name, including the pathname, of the contact force CTF file (intfor)
num_ families	integer	the number of adaptive remesh families in the file sequence. Will be one for a normal non-adaptive analysis
num_states	integer	the number of complete states in the file sequence
op2_name	string	the full name, including the pathname, of the Nastran OP2 file
pp_name	string	the full name, including the pathname, of the LS-PREPOST database file
ptf_name	string	the full name, including the pathname, of the complete state PTF/d3plot file
xtf_name	string	the full name, including the pathname, of the extra database XTF file
ztf_name	string	the full name, including the pathname, of the extra database ZTF file

Return type

object

Example

```
// Print the name of the PTF (d3plot) file of the current model and the number
of states
var info = GetModelInfo();
Print("PTF filename = " + info.ptf_name + "\n");
Print("Number of states = " + info.num_states + "\n");
// Print the name of the 3rd adaptive remesh PTF file in model 2
var info = GetModelInfo(2, 3);
Print("PTF filename = " + info.ptf_name + "\n");
```

ModelExists(model_id*[integer]*) [static]

Description

Checks whether a model exists in the database

Arguments

- **model_id** (integer)

Model number to check

Returns

boolean

Return type

Boolean

Example

```
// Check if model #2 exists
ModelExists(2);
```

SetCurrentModel(model_id*[integer]*) [static]

Description

Sets the current model in the JavaScript interface to model_id

At the start of script execution the current model is automatically set to the first active model in the database

Arguments

- **model_id** (integer)

Model number to be made current

Returns

boolean

Return type

Boolean

Example

```
// Make model #2 current
SetCurrentModel(2);
```

Parts

Functions and constants relating to Parts

Functions

- GetPartInfo(part_id*[integer]*)
- GetPid(type_code*[integer]*, item*[integer]*, state_id (optional)*[integer]*)

Details of functions

GetPartInfo(part_id*[integer]*) [static]

Description

Returns information about a part in the current model

Arguments

- **part_id** (integer)

Internal part number

Returns

Object with the following properties:

Name	Type	Description
alpha	integer	Part transparency (0-255)
blue	integer	Blue component of part colour (0-255)
green	integer	Green component of part colour (0-255)
include	integer	The include number part is in (0 if main file)
red	integer	Red component of part colour (0-255)
title	string	The part title

Return type

object

Example

```
// Return the title of the first part in the model
var info = GetPartInfo(1);
Print("Part title = " + info.title + "\n");
```

GetPid(type_code*[integer]*, item*[integer]*, state_id (optional)*[integer]*) [static]

Description

Returns the internal part id of internal <item> of type <type_code>

Arguments

- **type_code** (integer)

A valid part-based element type code (e.g. SOLID, SHELL)

- **item** (integer)

If +ve, the internal item number starting at 1. If -ve, the external label of the item. Internal numbers will be many times faster to process.

- **state_id (optional)** (integer)

State number to be used instead of the current state

Returns

integer

Return type

Number

Example

```
// Return the internal part index of the 27th internal shell
var a = GetPid(SHELL, 27);
// Return the external part label of the shell with label 123
var pid = GetLabel(PART, GetPid(SHELL, -123));
```

Selecting

Functions and constants relating to Selecting

Functions

- IsSelected(type_code*[integer]*, item*[integer]*)
- Pick(type_code*[integer]*, number*[integer]*)
- Select(type_code*[integer]*)

Details of functions

IsSelected(type_code*[integer]*, item*[integer]*) [static]

Description

Checks whether an item has been selected with Select()

Arguments

- **type_code** (integer)

The type of item to select (SOLID, etc.)

- **item** (integer)

If +ve, the internal item number starting at 1. If -ve, the external label of the item.

Returns

boolean

Return type

Boolean

Example

```
// Returns JS_TRUE if the 1st PART in the model was selected.
if(IsSelected(PART, 1))
{
    ....
}
```

Pick(type_code*[integer]*, number*[integer]*) [static]

Description

Allows the user to interactively pick a specified number of items

NOTE: If you are using the WINDOW type code, the function should be seen as "Pick item/model and return its WINDOW ID" i.e. if you try and use the pick function and click somewhere away from the model, the function will return null. On the other hand, if you click the model then it will return the WINDOW ID in which the model resides.

Arguments

- **type_code** (integer)

The type of item to select (SOLID, etc.)

- **number** (integer)

The number of items to pick.
- > 0 The internal indicies of the picked items are returned
- < 0 The external labels of the picked items are returned

Returns

Array of integers (or null if not picked)

Return type

Number

Example

```
// Pick 4 PARTS and return the internal index of each one in array (a)
var a = Pick(Type.PART, 4);
// Pick 3 NODES and return the external labels of each one in array (b)
var b = Pick(Type.NODE, -3);
if(b)
{
    Message("picked nodes " + b[0] + ", " + b[1] + " and " + b[2]);
}
else
{
    Message("cancelled picking");
}
```

Select(type_code*[integer]*) [static]

Description

Allows the user to interactively select items using the mouse or from a menu.

Arguments

- **type_code** (integer)

The type of item to select (SOLID, etc.)

Returns

integer, >0 the number of items selected, -1 user cancelled the operation, -2 model doesn’t contain any of the type requested

Return type

Number

Example

```
// Select PARTS interactively and return the number selected.
var a = Select(PART);
```

Sets

Functions and constants relating to Sets

Functions

- GetItemsInSet(set_type*[integer]*, set_id*[integer]*)
- GetSetInfo(set_type*[integer]*, set_id*[integer]*)

Details of functions

GetItemsInSet(set_type*[integer]*, set_id*[integer]*) [static]

Description

Returns an object containing the number of items in set <set_id> of set type <set_type> and also an array <list[ ]> of their internal indices. If there are no items in the set then false is returned.

Arguments

- **set_type** (integer)

A valid type code (SET_PART, etc.)

- **set_id** (integer)

The set id. If +ve, the internal number starting at 1. If -ve, the external label of the set. Internal numbers will be many times faster to process.

Returns

Object with the following properties:

Name	Type	Description
list	array of integers	Internal entity indices
nn	integer	Number of entities in list

Return type

object

Example

```
// Get a list of parts in the 5th SET_PART
if(a = GetItemsInSet(SET_PART, 5))
{
    var nparts = a.nn;
    for(var i=0; i<nparts; i++)
    {
        Message("Part: " + GetLabel(PART, a.list[i]));
    }
}
```

GetSetInfo(set_type*[integer]*, set_id*[integer]*) [static]

Description

Returns information about a set in the current model

Arguments

- **set_type** (integer)

A valid type code (SET_PART, etc.)

- **set_id** (integer)

The set id. If +ve, the internal number starting at 1. If -ve, the external label of the set. Internal numbers will be many times faster to process.

Returns

Object with the following properties:

Name	Type	Description
label	integer	The label of the set
name	string	The name of the set
nn	integer	Number of items in the set

Return type

object

Example

```
// Print the name of the first *SET_PART in the current model
var info = GetSetInfo(SET_PART, 1);
Print("Set name = " + info.name + "\n");
```

SharedConstants

Functions and constants relating to SharedConstants

SharedConstants constants

Constants for Dispose

Name	Description
DELETE	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Delete. Use Constant.DELETE instead [deprecated]
LEAVE	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Leave behind (eg don't delete). Use Constant.LEAVE instead [deprecated]

Constants for General

Name	Description
ALL	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. All of a category. Use Constant.ALL instead [deprecated]
GT	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Greater than (>). Use Constant.GT instead [deprecated]
GTEQ	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Greater than or equals (>=). Use Constant.GTEQ instead [deprecated]
LT	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Less than (<). Use Constant.LT instead [deprecated]
LTEQ	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Less than or equals (<=). Use Constant.LTEQ instead [deprecated]
MAX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Maximum value. Use Constant.MAX instead [deprecated]
MIN	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Minimum value. Use Constant.MIN instead [deprecated]
OFF	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Switch off. Use Constant.OFF instead [deprecated]
ON	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Switch on. Use Constant.ON instead [deprecated]

Constants for Item Type

Name	Description
BEAM	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Beams. Use Type.BEAM instead [deprecated]

BOLT	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Bolts. Use Type.BOLT instead [deprecated]
BWLD	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Beam spotwelds. Use Type.BWLD instead [deprecated]
CONN	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. All connection types. Use Type.CONX instead [deprecated]
CWLD	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. *CONSTRAINED_SPOTWELD spotwelds. Use Type.CWLD instead [deprecated]
DES	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Discrete Element Sphere. Use Type.DES instead [deprecated]
ELEM	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Generic elements. Use Type.ELEMENT instead [deprecated]
GWLD	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. *CONSTRAINED_GENERALIZED spotwelds. Use Type.GWLD instead [deprecated]
HSWA	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Hex spotweld assemblies. Use Type.HSWA instead [deprecated]
HWLD	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Hex (Solid) spotwelds. Use Type.HWLD instead [deprecated]
JOINT	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Joints. Use Type.JOINT instead [deprecated]
MASS	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Lumped masses. Use Type.MASS instead [deprecated]
MIG	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. MIG welds. Use Type.MIG instead [deprecated]
NODE	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Nodes. Use Type.NODE instead [deprecated]
NRB	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Nodal Rigid Bodies. Use Type.NRB instead [deprecated]
PART	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Parts. Use Type.PART instead [deprecated]
PRET	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Pretensioners. Use Type.PRETENSIONER instead [deprecated]
RBOLT	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Rigid bolts. Use Type.RBOLT instead [deprecated]
RETR	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Retractors. Use Type.RETRACTOR instead [deprecated]

SBELT	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Seatbelt elements. Use Type.SEATBELT instead [deprecated]
SBENT	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Seatbelt types generally. Use Type.SBENT instead [deprecated]
SECT	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. (Element) section definitions. Use Type.SECTION instead [deprecated]
SEGM	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Interface (contact, blast, etc) segment. Use Type.SEGMENT instead [deprecated]
SET_BEAM	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. *SET_BEAM sets. Use Type.SET_BEAM instead [deprecated]
SET_ DISCRETE	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. *SET_DISCRETE sets. Use Type.SET_DISCRETE instead [deprecated]
SET_NODE	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. *SET_NODE sets. Use Type.SET_NODE instead [deprecated]
SET_PART	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. *SET_PART sets. Use Type.SET_PART instead [deprecated]
SET_SHELL	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. *SET_SHELL sets. Use Type.SET_SHELL instead [deprecated]
SET_SOLID	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. *SET_SOLID sets. Use Type.SET_SOLID instead [deprecated]
SET_ TSHELL	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. *SET_TSHELL sets. Use Type.SET_TSHELL instead [deprecated]
SHELL	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Shells. Use Type.SHELL instead [deprecated]
SLIP	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Slip-rings. Use Type.SLIPRING instead [deprecated]
SOLID	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Solids. Use Type.SOLID instead [deprecated]
SPC	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Single Point Constraint. Use Type.SPC instead [deprecated]
SPH	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Smoothed Particle Hydrodynamics. Use Type.SPH instead [deprecated]
SPRING	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Springs (discrete elements). Use Type.SPRING instead [deprecated]
SURF	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Contact surfaces. Use Type.CONTACT instead [deprecated]

TSHELL	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Thick shells. Use Type.TSHELL instead [deprecated]
WALL	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Rigidwalls. Use Type.RIGIDWALL instead [deprecated]
WINDOW	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. D3PLOT window id. Use Type.WINDOW instead [deprecated]
XSEC	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Database cross-sections. Use Type.XSEC instead [deprecated]

Constants for Surface

Name	Description
BOTTOM	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Bottom shell surface. Use Constant.BOTTOM instead [deprecated]
MIDDLE	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Middle shell surface. Use Constant.MIDDLE instead [deprecated]
TOP	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Top shell surface. Use Constant.TOP instead [deprecated]

Constants for Tensor Array Subscripts

Name	Description
XX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. 0 for tensors. Use Constant.XX instead [deprecated]
XY	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. 3 for tensors (can also use YX as an alternative). Use Constant.XY instead [deprecated]
YY	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. 1 for tensors. Use Constant.YY instead [deprecated]
YZ	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. 4 for tensors (can also use ZY as an alternative). Use Constant.YZ instead [deprecated]
ZX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. 5 for tensors (can also use XZ as an alternative). Use Constant.ZX instead [deprecated]
ZZ	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. 2 for tensors. Use Constant.ZZ instead [deprecated]

Constants for Vector Array Subscripts

Name	Description
X	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. 0 for vectors. Use Constant.X instead [deprecated]

Y	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. 1 for vectors. Use Constant.Y instead [deprecated]
Z	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. 2 for vectors. Use Constant.Z instead [deprecated]

Ssh class

The Ssh class allows you to connect to a remote computer using ssh, scp and sftp commands. More...

The D3PLOT JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Member functions

- AuthenticateWithPassword(password*[string]*)
- AuthenticateWithPublicKey(passphrase (optional)*[string]*)
- Execute(data*[object]*)
- Get(remote*[string]*, local*[string]*)
- Put(remote*[string]*, local*[string]*)
- SftpGet(remote*[string]*, local*[string]*)
- SftpList(remote*[string]*)
- SftpMkdir(remote*[string]*, mode*[constant]*)
- SftpPut(remote*[string]*, local*[string]*)
- SftpRmdir(remote*[string]*)

Ssh constants

Constants for file bits

Name	Description
Ssh.SETGROUP_BIT	Set group bit
Ssh.SETUID_BIT	Set uid bit
Ssh.STICKY_BIT	sticky bit

Constants for file types

Name	Description
Ssh.DIRECTORY	Directory
Ssh.FILE	Regular file
Ssh.SOCKET	Socket
Ssh.SYMBOLIC_LINK	Symbolic link

Constants for permissions

Name	Description
Ssh.GROUP_EXECUTE	Group has execute permission
Ssh.GROUP_READ	Group has read permission
Ssh.GROUP_WRITE	Group has write permission
Ssh.OTHER_EXECUTE	Others have execute permission
Ssh.OTHER_READ	Others have read permission

Ssh.OTHER_WRITE	Others have write permission
Ssh.OWNER_EXECUTE	Owner has execute permission
Ssh.OWNER_READ	Owner has read permission
Ssh.OWNER_WRITE	Owner has write permission

Detailed Description

The Ssh class gives you simple functions to do secure connections to a remote computer using ssh. Oasys LS-DYNA Environment software is built with the OpenSSH library to support the ssh, scp and sftp protocols. The basic workflow is to create a connection using the Ssh constructor, authenticate the connection either by using a password and AuthenticateWithPassword or with a public key and AuthenticateWithPublicKey then the method Execute can be used to execute commands on the remote machine, the methods Get and Put can be used to copy files to and from the remote machine using scp, and the commands SftpGet, SftpList, SftpMkdir, SftpPut and SftpRmdir can be used to perform secure file transfer commands.
ssh uses a public and private key pair to do communication. The software uses RSA for the private and public keys and stores them in the files id_rsa and id_rsa.pub in the .oasys_ssh directory of your home directory (C:\Users\your.name\.oasys_ssh on Windows by default). A key length of 2048 bits is recommended. You keep your private key secure in your .oasys_ssh directory but the public key can be copied to the authorized_keys file on remote machines so that authentication can be done etc. The software also maintains fingerprints for the machines you connect to to ensure that you are connecting to the machine that you think you are. The first time you connect to a machine you are asked to confirm the remote machine is correct and the software stores the fingerprint for it in the known_hosts file in your .oasys_ssh directory. For second and subsequent connections the software checks the fingerprint of the remote machine against the one it has stored and will only connect if it matches.
When creating a new ssh connection to a remote machine and transferring files a small 'buffer' is required to transfer the data. The size of this buffer can be controlled using the Options.ssh_buffer_size property **before** the Ssh object is created.

Constructor

new Ssh(hostname*[string]*, username*[string]*)

Description

Create a new Ssh object for secure communication to a remote computer.

Arguments

- **hostname** (string)

The hostname of the machine that you want to connect to

- **username** (string)

The username on the machine that you want to connect to

Returns

Ssh object

Return type

Ssh

Example

To create a connection to machine "example" as user "username"

```
var s = new Ssh("example", "username");
```

Details of functions

AuthenticateWithPassword(password*[string]*)

Description

Authenticate the connection using password.

Arguments

- **password** (string)

The password for the username on the remote machine

Returns

no return value

Example

To prompt the user for a password and authenticate using it in SSH connection s:

```
var password = Window.GetPassword("Enter Password to connect", "Password");
s.AuthenticateWithPassword(password);
```

AuthenticateWithPublicKey(passphrase (optional)*[string]*)

Description

Authenticate the connection using your public key. Your public key from the file .oasys_ssh/id_rsa.pub must be in the file .oasys_ssh/authorized_keys on the remote machine.

Arguments

- **passphrase (optional)** (string)

The passphrase for authentication on the remote machine if required

Returns

no return value

Example

Authenticate using your public key in SSH connection s:

```
s.AuthenticateWithPublicKey();
```

Execute(data*[object]*)

Description

Execute a command in the ssh session and get the standard output and error streams.

Arguments

- **data** (object)

Execute data

Object has the following properties:

Name	Type	Description
arguments (optional)	Array of strings	The arguments to pass to the command

command	string	The command you want to run

Returns

Object with the following properties:

Name	Type	Description
status	integer	The exit code from the command
stderr	string	The standard error output from the command
stdout	string	The standard output from the command

Return type

object

Example

To run command "example.bat" with arguments "foo" and "bar" in SSH connection s:

```
var output = s.Execute( { command: 'example.bat', arguments: [ 'foo', 'bar' ]}
);
var text   = output.stdout;
var errors = output.stderr;
var ecode  = output.status;
```

Get(remote*[string]*, local*[string]*)

Description

Gets a file from the ssh connection using scp.

Arguments

- **remote** (string)

The path of the remote file to get

- **local** (string)

The path of the local file to write

Returns

no return value

Example

To get the remote file "/path/to/file.txt", creating local file "C:\path\to\file.txt" in SSH connection s:

```
s.Get("/path/to/file.txt", "C:\\path\\to\\file.txt");
```

Put(remote*[string]*, local*[string]*)

Description

Puts a file on the remote ssh connection using scp.

Arguments

- **remote** (string)

The path of the remote file to put

- **local** (string)

The path of the local file to read

Returns

no return value

Example

To put the local file "C:\path\to\file.txt" to remote file "/path/to/file.txt" in SSH connection s:

```
s.Put("/path/to/file.txt", "C:\\path\\to\\file.txt");
```

SftpGet(remote*[string]*, local*[string]*)

Description

Gets a file from the ssh connection using sftp.

Arguments

- **remote** (string)

The path of the remote file to get

- **local** (string)

The path of the local file to write

Returns

no return value

Example

To get the remote file "file.txt", creating local file "C:\path\to\file.txt" in SSH connection s using sftp:

```
s.SftpGet("file.txt", "C:\\path\\to\\file.txt");
```

SftpList(remote*[string]*)

Description

Gets a listing from the ssh connection using sftp.

Arguments

- **remote** (string)

The remote path to get the listing from

Returns

Array of objects. Each object contains the following information for a file/directory:

Name	Type	Description
atime	integer	Access time for the file (seconds since epoch)
gid	integer	The group ID
info	constant	Bitwise information for the file/directory. See the permissions, file types and file bits constants
mtime	integer	Modification time for the file (seconds since epoch)
name	string	The name of the file/directory
size	integer	The size of the file
uid	integer	The user ID

Return type

object

Example

To get listing from the the remote path "temp" in SSH connection s using sftp:

```
var listing = s.SftpList("temp");
for (l=0; l<listing.length; l++)
{
    Message(listing[l].name + ":" + listing[l].size;
}
```

SftpMkdir(remote*[string]*, mode*[constant]*)

Description

Creates a directory in the remote ssh connection using sftp.

Arguments

- **remote** (string)

The remote directory to create

- **mode** (constant)

The mode/permissions for the directory. See the permissions constants for details. Note that the user's file-creation mask (umask) value will also be taken into account when creating the directory.

Returns

true if successful, false if not

Return type

Boolean

Example

To create the remote path "temp" with, create, write and execute permissions for only the owner, in SSH connection s using sftp:

```
var success = s.SftpMkdir("temp", Ssh.OWNER_READ|Ssh.OWNER_WRITE|Ssh.OWNER_
EXECUTE);
```

SftpPut(remote*[string]*, local*[string]*)

Description

Puts a file on the remote ssh connection using sftp.

Arguments

- **remote** (string)

The path of the remote file to put

- **local** (string)

The path of the local file to read

Returns

no return value

Example

To put the local file "C:\path\to\file.txt" to remote file "file.txt" in SSH connection s:

```
s.SftpPut("file.txt", "C:\\path\\to\\file.txt");
```

SftpRmdir(remote*[string]*)

Description

Deletes a directory in the remote ssh connection using sftp. If this fails it is probably because the directory is not empty.

Arguments

- **remote** (string)

The remote directory to delete

Returns

true if successful, false if not

Return type

Boolean

Example

To delete the remote path "temp" in SSH connection s using sftp:

```
var success = s.SftpRmdir("temp");
```

States

Functions and constants relating to States

Functions

- GetTime(state_id (optional)*[integer]*)
- LockState(state_id*[integer]*)
- SetCurrentState(state_id*[integer]*)
- UnlockState(state_id*[integer]*)

Details of functions

GetTime(state_id (optional)*[integer]*) [static]

Description

Returns the analysis time of the current state, or that of <state_id> if defined

Arguments

- **state_id (optional)** (integer)

State number to use

Returns

real

Return type

Number

Example

```
// Get the time of the current state
var time = GetTime();
// Get the time of the first state
var time = GetTime(1);
```

LockState(state_id*[integer]*) [static]

Description

"Locks" any memory already allocated for data storage in <state_id>, preventing it from being reused by other states looking for memory in which to store data.

When dealing with large models it is normally the case that the amount of data to be processed far exceeds the amount of memory installed in the computer, meaning that it is not possible to store all data of interest in memory at the same time. Therefore D3PLOT tries to minimise the amount of data currently stored in memory by reusing the memory allocated previously for other states and/or data components. This process is called "scavenging" and the rules it uses when trying to decide from where to scavenge memory are, in order of descending preference:

1. Data from a different component in a different state
2. Data from this component in a different state
3. Data from an unused component in this state
4. If none of the above are available then allocate some fresh memory from the operating system

In most cases a Javascript will be working with one state at a time, so the problem of reusing memory in this state for purpose A when it is still required for purpose B will not arise. However if, for example, you are writing a script that compares data from this state and the previous one inside a loop it is possible that "churning" could arise from the sequence:

.

For each state

GetData in state N Scavenges memory from state N-1 to store the data for state N

GetData in state N-1 Scavenges memory from state N to store data for state N-1

.

In this example the script would probably run incredibly slowly as each GetData() call would have to reread data from disk into the newly scavenged memory, so you would end up with <#elements * 2> disk reads of all the data for this component and element type. The same would be true if PutUbinData() or GetUbinData() were used as both of these require the data to be "put" or "got" to exist in memory, requiring that memory to be obtained from somewhere.

By "locking" states **N** and **N-1** in this example you would force D3PLOT to allocate enough memory to hold both data vectors in memory at the same time, and the script would run <#elements * 2> times faster. For a model with 1,000,000 elements this might reduce the run-time from months to seconds!

Clearly states should not be "locked" unnecessarily or, more importantly, left "locked" when there is no longer any need for the data they contain, since this will lead to a significant build-up of memory usage. Therefore states can be unlocked in three ways:

- Explicitly by using the Javascript function UnlockState()
- Implicitly by using the Javascript function SetCurrentState(), which unlocks all states except the current one
- Implicitly by exiting the Javascript, as normal (interactive or batch) D3PLOT usage will implicitly unlock all but the current state.

To summarise: this function is likely to be needed only when you are performing repeated "gets" and/or "puts" of data to and from more than one state.

Locking and unlocking states takes place in the current model only, and has no effect on states in any other model.

Arguments

- **state_id** (integer)

State number to lock

Returns

boolean

Return type

Boolean

Example

```
// Lock data in state #13
LockState(13);
```

SetCurrentState(state_id*[integer]*) [static]

Description

Sets the current state for the JavaScript interface to state_id

This is the state used for all the "get" and "put" functions which handle model-related data. If the optional state_id argument in a get/put function call is used then that state is used instead for the duration of that call, but this current state is not changed.

The current state is a property of the current model, in other words each model has its own, separate, current state. For all models this defaults to state #1 (if present).

Setting the current state in model i has no effect on the current state in any other model.

Arguments

- **state_id** (integer)

State number to make current

Returns

boolean

Return type

Boolean

Example

```
// Make state #27 current
SetCurrentState(27);
```

UnlockState(state_id*[integer]*) [static]

Description

"Unlocks" this state for the purposes of memory scavenging, making any data vectors within it eligible for reuse by other states looking for memory

Please see the documentation on LockState() for a description of what this function does and when it might be needed.

Arguments

- **state_id** (integer)

State number to unlock

Returns

boolean

Return type

Boolean

Example

```
// Unlock data in state #13
UnlockState(13);
```

UserComponents

Functions and constants relating to UserComponents

Functions

- CreateUbinComponent(component_name*[string]*, component_type*[integer]*, data_type*[integer]*, if_existing*[integer]*, dispose (optional)*[integer]*, location (optional)*[integer or string]*)
- DeleteUbinComponent(handle*[integer]*)
- GetUbinData(handle*[integer]*, item_type*[integer]*, item*[integer]*, int_pt*[object | integer]*, state_id (optional)*[integer]*)
- LocateUbinComponent(component_name*[string]*)
- PutUbinData(handle*[integer]*, item_type*[integer]*, item*[integer]*, int_pt*[object | integer]*, data*[real|array of reals]*, state_id (optional)*[integer]*)

UserComponents constants

Name	Description
UBMS	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Beam scalar. Use Component.UBMS instead [deprecated]
UBMV	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Beam vector. Use Component.UBMV instead [deprecated]
UNOS	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Node scalar. Use Component.UNOS instead [deprecated]
UNOV	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Node vector. Use Component.UNOV instead [deprecated]
USSS	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Solid and shell scalar. Use Component.USSS instead [deprecated]
USST	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Solid and shell tensor. Use Component.USST instead [deprecated]

Constants for Component Type

Name	Description
U_BEAM	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. User-defined beam component. Use Component.BEAM instead [deprecated]
U_NODE	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. User-defined nodal component. Use Component.NODE instead [deprecated]
U_OTHR	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. User-defined other (LSDA) component. Use Component.OTHER instead [deprecated]
U_SOSH	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. User-defined solid, shell and thick shell component. Use Component.SOLID_SHELL_TSHELL instead [deprecated]

Constants for Data Type

Name	Description
U_SCALAR	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Scalar data (1 value). Use Component.SCALAR instead [deprecated]
U_TENSOR	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Tensor data (6 values). Use Component.TENSOR instead [deprecated]
U_VECTOR	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Vector data (3 values). Use Component.VECTOR instead [deprecated]

Constants for Existing

Name	Description
RENAME	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Rename. Use Component.RENAME instead [deprecated]
REPLACE	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. Replace. Use Component.REPLACE instead [deprecated]

Constants for Location

Name	Description
IN_CORE	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. held in memory. Use Component.IN_CORE instead [deprecated]

Details of functions

CreateUbinComponent(component_name*[string]*, component_type*[integer]*, data_type*[integer]*, if_existing*[integer]*, dispose (optional)*[integer]*, location (optional)*[integer or string]*) [static]

Description

Create a new user-defined binary (UBIN) component

Note that user-defined components are "program wide", so once created the data "slots" exist in all models. Data values that are not populated will return a value of zero.

Arguments

- **component_name** (string)

A name for this component, up to 30 characters long. If the name is not unique, D3PLOT's behaviour will depend on the value of the 'if_existing' argument to this function.

- **component_type** (integer)

One of the constants
- U_NODE for nodal data
- U_SOSH for solid, shell and thick shell data
- U_BEAM for beam data
- U_OTHR for LSDA (Other) data

User-defined components must fall into one of these four categories. It is not possible to have a component of a given name that contains data for more than one of these types.

- **data_type** (integer)

One of the constants
- U_SCALAR for scalar data (any type)

- U_VECTOR for vector data (U_NODE, U_BEAM and U_OTHR only)
- U_TENSOR for tensor data (U_SOSH only)
- U_OTHR for LSDA (Other) data

Choose the data type that matches the information you want to store.

- **if_existing** (integer)

Action to take if UBIN component 'component_name' already exists.

One of the constants

- REPLACE deletes the existing UBIN component, replacing it with this definition. This means that any existing data for the user-defined component of this name is deleted and the component is re-initialised.
- RENAME changes the 'component_name' argument of this function call by adding a suffix to make it unique, so the existing component of this name (and data) will be left unchanged and the new one will not clash with it.

- **dispose (optional)** (integer)

What to do with the ".ubd" files when the model is closed or D3PLOT exits.

One of the constants

- LEAVE (default) will leave any ".ubd" files on disk so that they are available for any future D3PLOT sessions.
- DELETE will delete these files when then model is closed or D3PLOT exits.

If this argument is ommitted or set to zero then LEAVE behaviour is used. However alternative default behaviour may be specified by setting the preference

d3plot*ubd_file_dispose: to **LEAVE** or **DELETE**

- **location (optional)** (integer or string)

Specify where the data for this component is to be stored, one of

- A valid <pathname> .ubd files will be written to this directory instead of the original analysis. This will usually be a better solution than the alternative options of keeping data "in core" since it allows D3PLOT memory management to operate normally, writing data to disk if space is needed in memory. The directory <pathname> must exist, and you must have write permission to it.
- **JOBDIR**(<pathname>) the path of the directory containing the results, in other words the default location for the files. However you can append a further <pathname> to this in order to specify a directory relative to JOBDIR, for example:
  - **JOBDIR/..** Means the directory above the current results
  - **JOBDIR/../../my_results** Means two directories above, in the sub-directory 'my_results'
- IN_CORE stipulates that this component's data will always be held in memory, and will never be written to disk. This solves the problem of data files being in read-only directories since no .ubd files are written. However it also means that D3PLOT will not dump data for currently unused states to disk, meaning that you may run out of memory if you generate too much data in your JavaScripts.

If IN_CORE is used the value of 'dispose' above is ignored.

If this argument is omitted then the default behaviour of creating .ubd files in the same directory as the analysis database files will be used. However an alternative default directory may also be specified by the preference:

d3plot*ubd_file_location: <pathname> or **IN_CORE** or **JOBDIR**(<pathname>)

If both <location> and this preferenceare defined then <location> in this function call takes precedence.

Notes on pathnames:

1. On Windows platforms forward slash / and backslash \ can be used interchangeably in pathnames. On Linux platforms you must use forward slash / only, so in a multi-system environment it is recommended that you use forward slash syntax only.
2. If <pathname> contains white space then you must enclose the whole string in "...", for example "C:\my results".

Returns

integer handle for the newly created component that should be used in subsequent UBIN processing function calls. This handle should be regarded as private data and not modified in any way. In addition, if a UBIN component is created and then recreated and over-written in a script (**if_existing = REPLACE**) the handle from each call may be different - don't assume that is has not changed.

Return type

Number

Example

```
// Create a component for nodal scalar data
var handle_1 = CreateUbinComponent("My nodal data", U_NODE, U_SCALAR, REPLACE);
// Create a tensor component for solid, shell and thick shell data
var handle_2 = CreateUbinComponent("My shell tensor data", U_SOSH, U_TENSOR,
REPLACE);
```

DeleteUbinComponent(handle*[integer]*) [static]

Description

Deletes an existing UBIN component handle. The component is deleted from memory, and any associated .ubd files cached on disk are also deleted.

If this succeeds it returns JS_TRUE, otherwise JS_FALSE.

Arguments

- **handle** (integer)

The handle of an existing UBIN component

Returns

boolean

Return type

Boolean

Example

```
// Delete the UBIN component handle_1
if(!DeleteUbinComponent(handle_1))
{
    ...deal with failure...
}
```

GetUbinData(handle*[integer]*, item_type*[integer]*, item*[integer]*, int_pt*[object | integer]*, state_id (optional)*[integer]*) [static]

Description

Retrieves data for type/item from a UBIN component.

If the data has not previoulsy been written, values of 0.0 will be returned.

Arguments

- **handle** (integer)

The handle of an existing UBIN component as returned by CreateUbinComponent().

- **item_type** (integer)

An item type constant, NODE, SOLID, SHELL, etc. This must match the underlying type of the UBIN component, thus NODE for components of type U_NODE, and so on. It is illegal to attempt to store data for a type that does not match the underlying UBIN component type thus, for example, you cannot store NODE data for a U_SOSH component.

- **item** (integer)

If +ve, the internal item number starting at 1. If -ve, the external label of the item. Internal numbers will be many times faster to process.

- **int_pt** (object) | integer

Integration point: must be a +ve layer number (lowest = 1).

Or zero for item type / data component combinations that do not consider integration points in this context. (for example nodal displacements or beam forces).

Or, for fully integrated elements with on plan integration points, an object with properties "ip" and "op". For a further explanation see Defining the Integration point argument in GetData().

"Top", "Middle" and "Bottom" are not allowed in this context since "middle" is not directly readable in cases with an even number of points.

A value of 1 should normally be used for solid elements.

Note, from D3PLOT 11.0 onwards, the order of the integration points for SHELLS and TSHELLS is <int_pnt> 1->n: BOTTOM->TOP surface (so long as a ZTF file is present). See Section 13.8.2.2.

Prior to this they were in the order of the integration points output by Ansys LS-DYNA, e.g. for <maxint>=3 <int_pnt> 1 was the MIDDLE surface, <int_pnt> 2 was the BOTTOM surface and <int_pnt> 3 was the TOP surface.

Object has the following properties:

Name	Type	Description
ip	integer	Through thickness integration point.
op (optional)	integer	On plan integration point. Defaults to the first one.

- **state_id (optional)** (integer)

State number to be used. If omitted, the current state is used.

Returns

real|Array of reals

Return type

Number

Example

```
// Retrieve an array of tensor data for solid #27, which implies that the UBIN
data
// component <handle_1> is of type U_SOSH, and that its data type is U_TENSOR.
dvec = GetUbinData(handle_1, SOLID, 27, 1);
sxx = dvec[0];
szx = dvec[5];
// Retrieve the scalar value of node #17, in state <istate>.
// This implies that the UBIN component <handle_2> is of type U_NODE and its
data is U_SCALAR.
nval = GetUbinData(handle_2, NODE, 17, 0, istate);
```

LocateUbinComponent(component_name*[string]*) [static]

Description

Locates an existing UBIN component by name and returns its handle. This is useful when a previous run has created a UBIN component and this script wishes to work with it.

'component_name' is not case-sensitive, but an exact character match is required, so embedded white space is significant.

If the lookup succeeds this function returns an object with with properties about the component, if it fails it returns the value JS_FALSE.

Arguments

- **component_name** (string)

A name to search for, a character string up to 30 characters long. Component names are not case-sensitive, but searching only succeeds if an exact match is found.

Returns

Object with the following properties:

Name	Type	Description
ctype	integer	the component type, U_NODE, U_SOSH, U_BEAM or U_OTHR
dtype	integer	the data type, U_SCALAR, U_VECTOR or U_TENSOR
handle	integer	the integer handle of the UBIN component

Return type

object

Example

```
// Look for component "My nodal data" and put the result of a successful lookup
in object 'udata'.
if(udata = LocateUbinComponent("My nodal data"))
{
    handle = udata.handle;
    ...
}
else
{
    ... deal with failure
}
```

PutUbinData(handle*[integer]*, item_type*[integer]*, item*[integer]*, int_pt*[object | integer]*, data*[real|array of reals]*, state_id (optional)*[integer]*) [static]

Description

Stores data for type/item in a UBIN component handle.

This will overwrite any existing data in that "slot", which will be lost.

Arguments

- **handle** (integer)

The handle of an existing UBIN component as returned by CreateUbinComponent().

- **item_type** (integer)

An item type constant, NODE, SOLID, SHELL etc. This must match the underlying type of the UBIN component, thus NODE for components of type U_NODE, and so on. It is illegal to attempt to store data for a type that does not match the underlying UBIN component type thus, for example, you cannot store NODE data for a U_SOSH component.

- **item** (integer)

If +ve, the internal item number starting at 1. If -ve, the external label of the item. Internal numbers will be many times faster to process.

- **int_pt** (object) | integer

Integration point: must be a +ve layer number (lowest = 1)

Or zero for item type / data component combinations that do not consider integration points in this context (for example nodal displacements or beam forces).

Or, for fully integrated elements with on plan integration points, an object with properties "ip" and "op". For a further explanation see Defining the Integration point argument in GetData().

"Top", "Middle" and "Bottom" are not allowed in this context since "middle" is not directly readable in cases with an even number of points.

A value of 1 should normally be used for solid elements.

Note from D3PLOT 11.0 onwards, the order of the integration points for SHELLS and TSHELLS is <int_pnt> 1->n: BOTTOM->TOP surface (so long as a ZTF file is present). See Section 13.8.2.2.

Prior to this they were in the order of the integration points output by Ansys LS-DYNA, e.g. for <maxint>=3 <int_pnt> 1 was the MIDDLE surface, <int_pnt> 2 was the BOTTOM surface and <int_pnt> 3 was the TOP surface.

Object has the following properties:

Name	Type	Description
ip	integer	Through thickness integration point.
op (optional)	integer	On plan integration point. Defaults to the first one.

- **data** (real|array of reals)

The data to be stored. Its format depends on the "data type" of the component:

U_SCALAR: Scalar or array of length >=1

U_VECTOR: Array of length >= 3

U_TENSOR: Array of length >= 6

The alignment of array members should be as follows:

Vector: [X, Y, Z]

Tensor: [XX, YY, ZZ, XY, YZ, ZX]

- **state_id (optional)** (integer)

State number to be used. If omitted, the current state is used.

Returns

boolean

Return type

Boolean

Example

```
// Write an array of tensor data for solid #27, which implies that the UBIN data
component <handle_1>
// is of type U_SOSH,and that its data type is U_TENSOR.
dvec = new Array(6);
dvec[XX] = sxx;
dvec[YZ] = syz;
PutUbinData(handle_1, SOLID, 27, 1, dvec);
// Write the scalar value 19.5 for node #17, in state <istate>.
// This implies that the UBIN component <handle_2> is of type U_NODE and its
data is U_SCALAR.
PutUbinData(handle_2, NODE, 17, 0, 19.5, istate);
```

Utils class

The Utils class contains various useful utility functions. More...

The D3PLOT JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Ascii85Decode(encoded*[string]*)
- Ascii85Encode(data*[ArrayBuffer]*, length (optional)*[integer]*)
- Build()
- CallPromiseHandlers()
- CheckinLicense(feature*[string]*)
- CheckoutLicense(feature*[string]*)
- GarbageCollect()
- HTMLBrowser()
- HiResTimer()
- PdfReader()
- SHA256(filename*[string]*)
- SHA512(filename*[string]*)
- Terminate()
- TimerResolution()
- UUID()
- Version()

Detailed Description

The Utils class is used to provide various useful functions.

Details of functions

Ascii85Decode(encoded*[string]*) [static]

Description

Decodes an ASCII85 encoded string. See Utils.Ascii85Encode() for details on the method.

Arguments

- **encoded** (string)

An ASCII85 encoded string

Returns

ArrayBuffer object

Return type

ArrayBuffer

Example

To decode an ASCII85 encoded string:

```
var decoded = Utils.Ascii85Decode(encoded);
```

Ascii85Encode(data*[ArrayBuffer]*, length (optional)*[integer]*) [static]

Description

Encodes an ASCII85 encoded string. This enables binary data to be represented by ASCII characters using five ASCII characters to represent four bytes of binary data (making the encoded size 1/4 larger than the original). By doing this binary data can be stored in JavaScript strings. Note that the method used by D3PLOT to encode and decode strings differs from the standard ASCII85 encoding as that uses the ASCII characters ", ’ and \ which cannot be used in JavaScript strings as they have special meanings. The method in D3PLOT uses
0-84 are !-u (ASCII codes 33-117) (i.e. 33 is added to it) with the following exceptions
v is used instead of " (ASCII code 118 instead of 34)
w is used instead of ’ (ASCII code 119 instead of 39)
x is used instead of \ (ASCII code 120 instead of 92)
If all five digits are 0 they are represented by a single character z instead of !!!!!

Arguments

- **data** (ArrayBuffer)

ArrayBuffer containing the data

- **length (optional)** (integer)

Length of data in array buffer to encode. If omitted the whole array buffer will be encoded

Returns

string

Return type

String

Example

To encode ArrayBuffer data:

```
var encoded = Utils.Ascii85Encode(data);
```

Build() [static]

Description

Returns the build number

Arguments

No arguments

Returns

integer

Return type

Number

Example

To get the current build number

```
var build = Utils.Build();
```

CallPromiseHandlers() [static]

Description

Manually call any promise handlers/callbacks in the job queue

Arguments

No arguments

Returns

no return value

Example

To run any queued promise handlers/callbacks:

```
Utils.CallPromiseHandlers();
```

CheckinLicense(feature*[string]*) [static]

Description

Checks a license for a feature back in

Arguments

- **feature** (string)

feature to check license back in for

Returns

no return value

Example

To check in a license for "EXAMPLE":

```
Utils.CheckinLicense("EXAMPLE");
```

CheckoutLicense(feature*[string]*) [static]

Description

Checks out a license for a feature

Arguments

- **feature** (string)

feature to check license for

Returns

true if license available, false if not

Return type

Boolean

Example

To checkout a license for "EXAMPLE":

```
var got = Utils.CheckoutLicense("EXAMPLE");
if (got == false) Exit();
```

GarbageCollect() [static]

Description

Forces garbage collection to be done. This should not normally need to be called but in exceptional circumstances it can be called to ensure that garbage collection is done to return memory.

Arguments

No arguments

Returns

no return value

Example

To force garbage collection to be done:

```
Utils.GarbageCollect();
```

HTMLBrowser() [static]

Description

Returns the path to the default HTML browser

Arguments

No arguments

Returns

string of the path

Return type

String

Example

To get path to the default HTML browser

```
var path = Utils.HTMLBrowser();
```

HiResTimer() [static]

Description

A high resolution timer that can be used to time how long things take. The first time this is called the timer will start and return 0. Subsequent calls will return the time in nanoseconds since the first call. Note that the timer will almost certainly not have 1 nanosecond precision but, depending on the platform, should should have a resolution of at least 1 microsecond. The resolution can be found by using Utils.TimerResolution()

Arguments

No arguments

Returns

number

Return type

number

Example

To time how long something takes to nanosecond precision:

```
var start = Utils.HiResTimer();
do something that takes some time...
var end = Utils.HiResTimer();
Message("it took " + (end-start) + "nanoseconds");
```

PdfReader() [static]

Description

Returns the path to the executable of the default pdf reader

Arguments

No arguments

Returns

string of the path

Return type

String

Example

To get path to the default pdf reader

```
var path = Utils.PdfReader();
```

SHA256(filename*[string]*) [static]

Description

Create a SHA-256 hash for a file

Arguments

- **filename** (string)

File to calculate the hash for

Returns

string

Return type

String

Example

To generate the hash for file "example.txt":

```
var sha256 = Utils.SHA256("example.txt");
```

SHA512(filename*[string]*) [static]

Description

Create a SHA-512 hash for a file

Arguments

- **filename** (string)

File to calculate the hash for

Returns

string

Return type

String

Example

To generate the hash for file "example.txt":

```
var sha512 = Utils.SHA512("example.txt");
```

Terminate() [static]

Description

Terminates the program. Note that this **terminates the entire program**, not the script. To terminate the script use Exit() instead

Arguments

No arguments

Returns

No return value

Example

To terminate the program:

```
Utils.Terminate();
```

TimerResolution() [static]

Description

Returns the resolution (precision) of the Utils.HiResTimer() timer in nanoseconds

Arguments

No arguments

Returns

number

Return type

number

Example

To find the resolution of the timer in nanoseconds:

```
var resolution = Utils.TimerResolution();
```

UUID() [static]

Description

Create an UUID (Universally Unique Identifier)

Arguments

No arguments

Returns

string

Return type

String

Example

To generate an UUID:

```
var uuid = Utils.UUID();
```

Version() [static]

Description

Returns the version number

Arguments

No arguments

Returns

real

Return type

Number

Example

To get the current version number

```
var version = Utils.Version();
```

Visibility

Functions and constants relating to Visibility

Functions

- Blank(type_code*[integer]*, item*[integer or array of integers or string]*, window_id (optional)*[integer]*)
- IsBlanked(type_code*[integer]*, item*[integer]*, window_id (optional)*[integer]*)
- IsDeleted(type_code*[integer]*, item*[integer]*, state_id (optional)*[integer]*)
- IsVisible(type_code*[integer]*, item*[integer]*, window_id*[integer]*, state_id (optional)*[integer]*)
- NumDeleted(type_code*[integer]*, state_id (optional)*[integer]*)
- Unblank(type_code*[integer]*, item*[integer or array of integers or string]*, window_id (optional)*[integer]*)

Details of functions

Blank(type_code*[integer]*, item*[integer or array of integers or string]*, window_id (optional)*[integer]*) [static]

Description

Blank an item

Arguments

- **type_code** (integer)

The type of item to check (SOLID, PART etc.) Note: If <item> is "ALL_DEL" (all deleted elements), only element types are acceptable.

- **item** (integer or array of integers or string)

If +ve, the internal item number starting at 1. If -ve, the external label of the item. It can also be an array of items (index/label) or a string indicating various items ("ALL" for all items of the type and "ALL_DEL" for deleted items of the type.)

- **window_id (optional)** (integer)

A window id. If defined then the item is blanked in that window. If not defined or set 0 to then the item is blanked in all windows.

Returns

No return value

Example

```
// Blanks the 1st PART in the current model in all windows
Blank(PART, 1);
// Blanks the 1st PART in the current model in window 2
Blank(PART, 1, 2);
// Blanks the 1st PART in the current model in all windows
Blank(PART, 1, 0);
// Blanks all PARTs in window 2
Blank(PART, "ALL", 2);
// Blanks everything in window 2
Blank(MODEL, "ALL", 2);
// Blanks all SHELLs specified in the array shell_list in window 1
Blank(SHELL, shell_list, 1);
```

IsBlanked(type_code*[integer]*, item*[integer]*, window_id (optional)*[integer]*) [static]

Description

Checks whether an item is currently blanked. If the type is PART then this function will only return true if all elements of the PART are currently blanked. If the PART is empty this returns false

Arguments

- **type_code** (integer)

The type of item to check (SOLID, etc.)

- **item** (integer)

If +ve, the internal item number starting at 1. If -ve, the external label of the item.

- **window_id (optional)** (integer)

A window id. If defined then the function will return true if the item is blanked in that window. If not defined or set to then the function returns true if it is blanked in any window.

Returns

boolean

Return type

Boolean

Example

```
// Return true if the 1st SHELL in the model is blanked in any window
if(IsBlanked(SHELL, 1))
{
    ....
}
// Return true if the 1st SHELL in the model is blanked in window 2
if(IsBlanked(SHELL, 1, 2))
{
    ....
}
// Return true if the 1st SHELL in the model is blanked in any window
if(IsBlanked(SHELL, 1, 0))
{
    ....
}
```

IsDeleted(type_code*[integer]*, item*[integer]*, state_id (optional)*[integer]*) [static]

Description

Checks whether an item is currently deleted. If the type is PART then this function will only return true if all the elements of the PART are currently deleted

The deleted status is computed as follows:

- Part-based elements: Ansys LS-DYNA reports the deletion status for part-based elements (but not DISCRETE or 1d SEATBELT elements) which have failed according to the failure criteria of their deletion model.
  Reincarnation of dead elements is possible: *DEFINE_CONSTRUCTION_STAGES will result in an inactive element being marked as deleted, and it will be "undeleted" if that stage becomes active later on in the analysis.
- Parts themselves: Ansys LS-DYNA does not delete parts as such. A deformable part in which all elements have been deleted is removed from the calculation, but this removal is not reported in the results database. D3PLOT considers a part to be deleted if it has no elements, or all of its elements are marked as deleted. Note that a rigid part with no elements is a perfectly legitimate- if unusual - construct in Ansys LS-DYNA.
- Nodes: Ansys LS-DYNA does not delete nodes, but nodes with no structural mass are removed from the calculation. However this removal is not reported in the results database. D3PLOT considers a node to be deleted if all the elements to which it is attached are themselves deleted. Remember that D3PLOT does not "know about" all possible connections to a node, for example it may be an extra node on a rigid body, in a rigid part set, or constrained in some other obscure way. Therefore the test "deleted if all attached nodes are deleted" may give false positives and should not be considered definitive.

Arguments

- **type_code** (integer)

This function only supports the following type codes. PART, NODE, SOLID, BEAM, TSHELL, SPH, DES

- **item** (integer)

If +ve, the internal item number starting at 1. If -ve, the external label of the item.

- **state_id (optional)** (integer)

A valid state id. If omitted the current state will be used.

Returns

boolean

Return type

Boolean

Example

```
// Returns true if the 1st SHELL in the model has been deleted
if(IsDeleted(SHELL, 1))
{
    ....
}
```

IsVisible(type_code*[integer]*, item*[integer]*, window_id*[integer]*, state_id (optional)*[integer]*) [static]

Description

Checks whether an item is currently visible.

An item is considered "visible" if the following conditions are all true:

1. Not blanked
2. The visibility switch is on for type_code
3. Is not empty, if type is PART
4. The item has not been deleted in the current state if the type is an element

Arguments

- **type_code** (integer)

This function only supports the following type codes. PART, NODE, SOLID, BEAM, TSHELL, SPH, DES

- **item** (integer)

If +ve, the internal item number starting at 1. If -ve, the external label of the item.

- **window_id** (integer)

A valid window id

- **state_id (optional)** (integer)

A valid state id. If omitted the current state will be used.

Returns

boolean

Return type

Boolean

Example

```
// Returns true if the 1st SHELL in the model in the first window is visible
if(IsVisible(SHELL, 1, 1))
{
    ....
}
```

NumDeleted(type_code*[integer]*, state_id (optional)*[integer]*) [static]

Description

Gets the number of deleted elements or segments

Arguments

- **type_code** (integer)

The type of item to check. Only accepts ELEM for #elements or SEGM for #segments.

- **state_id (optional)** (integer)

A state id. If defined then the number of deleted items is calculated for that state. If not defined or set to 0 then the number of items is calculated for the current state.

Returns

integer

Return type

Number

Example

```
// Number of deleted elements in current state of the current model
var a = NumDeleted(ELEM);
// Number of deleted elements in state 3 of the current model
var b = NumDeleted(ELEM, 3);
// Number of deleted segments in state 5 of the current model
var c = NumDeleted(SEGM, 5);
```

Unblank(type_code*[integer]*, item*[integer or array of integers or string]*, window_id (optional)*[integer]*) [static]

Description

Unblank an item

Arguments

- **type_code** (integer)

The type of item to check (SOLID, PART etc.) Note: If <item> is "ALL_DEL" (all deleted elements), only element types are acceptable.

- **item** (integer or array of integers or string)

If +ve, the internal item number starting at 1. If -ve, the external label of the item. It can also be an array of items (index/label) or a string indicating various items ("ALL" for all items of the type and "ALL_DEL" for deleted items of the type.)

- **window_id (optional)** (integer)

A window id. If defined then the item is unblanked in that window. If not defined or set to 0 then the item is unblanked in all windows.

Returns

No return value

Example

```
// Unblanks the 1st PART in the current model in all windows
Unblank(PART, 1);
// Unblanks the 1st PART in the current model in window 2
Unblank(PART, 1, 2);
// Unblanks the 1st PART in the current model in all windows
Unblank(PART, 1, 0);
// Unblanks all PARTs in window 2
Unblank(PART, "ALL", 2);
// Unblanks everything in window 2
Unblank(MODEL, "ALL", 2);
// Unblanks all SOLIDs specified in the array solid_list in window 1
Unblank(SOLID, solid_list, 1);
```

Windows

Functions and constants relating to Windows

Functions

- CreateWindow(model_list*[Array of integers|integer]*)
- DeleteWindow(window_list*[Array of numbers|number]*, dispose_flag (optional)*[integer]*)
- GetWindowFrame(window_id*[integer]*)
- GetWindowMaxFrame(window_id*[integer]*)
- GetWindowModels(window_id*[integer]*)
- SetWindowActive(window_id*[integer]*, active_flag*[integer]*)
- SetWindowFrame(window_id*[integer]*, frame_number*[integer]*)

Details of functions

CreateWindow(model_list*[Array of integers|integer]*) [static]

Description

Creates a new window containing one or more models contained in model_list

Arguments

- **model_list** (Array of integers|integer)

Model number(s). Can be a single model number, an array of model numbers or the constant ALL

Returns

boolean

Return type

Boolean

Example

```
// Create a new window containing models #2 and #3
var a = new Array(2, 3);
CreateWindow(a);
// Create a new window containing model #6
CreateWindow(6);
// Create a new window containing all currently active models
CreateWindow(ALL);
```

DeleteWindow(window_list*[Array of numbers|number]*, dispose_flag (optional)*[integer]*) [static]

Description

Deletes one or more windows in window_list, dealing with "orphaned" models according to dispose_flag.

WARNING

- D3PLOT does not permit gaps in window numbering, therefore when a window is deleted any windows higher than this are renumbered downwards to fill the gap.
- However D3PLOT does ***not*** renumber models following the deletion of preceding ones. Deleted model ids simply become "inactive".

This means that following a window deletion operation:

- The total number of windows will change.
- Any window ids above those deleted will have been renumbered downwards.
- If any orphan models were deleted these models will now be inactive.
- If the current Javascript model has been deleted then the "current" model pointer will be reset to the first active model, or <undefined> if there are no such models.

Therefore if a script is to continue execution after a window deletion operation it is prudent to ensure that any "current" user-defined variables in the Javascript are reset to sensible values.

Arguments

- **window_list** (Array of numbers|number)

Window numbers. Can be a single window number, an array of window numbers or the constant ALL

- **dispose_flag (optional)** (integer)

LEAVE (default) leaves orphaned models in the database or DELETE deletes orphaned models

Returns

boolean

Return type

Boolean

Example

```
// Delete windows #2 and #3 leaving any orphaned models in the database
var a = new Array(2, 3);
DeleteWindow(a);
// Delete window #6, also deleting any orphaned models
DeleteWindow(6, DELETE);
```

GetWindowFrame(window_id*[integer]*) [static]

Description

Returns the current "frame" in window_id

See the notes in GetWindowMaxFrame() on how frame number relates to state number

Arguments

- **window_id** (integer)

Window number or ALL. Specifies the window(s) to have the frame number set

Returns

integer

Return type

Number

Example

```
// Get the current frame of window #1
var a = GetWindowFrame(1);
```

GetWindowMaxFrame(window_id*[integer]*) [static]

Description

Returns the highest "frame" number in window_id

"Frame" number is usually the same as state number, but there are a few situations when this is not the case:

- Eigenvalue analyses. Each state is animated though *#frames* between +/-180 degrees phase angle
- Nastran-derived static analyses. Each loadcase is likewise animated through *#frames*
- Transient analyses that are being interpolated by time, giving (endtime / time interval) frames

In all cases animating a window results in it cycling through frames 1 to max *#frames*.

Arguments

- **window_id** (integer)

Window number

Returns

integer

Return type

Number

Example

```
// Get the highest frame of number in window #2
var a = GetWindowMaxFrame(2);
```

GetWindowModels(window_id*[integer]*) [static]

Description

Returns the model number(s) in window_id

Every active window in D3PLOT must have at least one model, but may have any number

Arguments

- **window_id** (integer)

Window number

Returns

Object with the following properties:

Name	Type	Description

list	Array of integers	List of model numbers
nm	integer	the number of models in the window

Return type

object

Example

```
// Get list of model numbers in window #1
var a = GetWindowModels(1);
for(i=0; i<a.nm; i++)
{
    Message("Model " + a.list[i] + " in window 1");
}
```

SetWindowActive(window_id*[integer]*, active_flag*[integer]*) [static]

Description

Set the "active" flag on a window.

When more than one window is in use it is convenient to be able to operate on a group of "active" windows with a single command in the JavaScript, rather than having to loop over selected windows each time, and this function provides that capability. This activity status is used solely within the Javascript interface and does not have any bearing upon or connection with the Wn "tabs" used in the graphical userinterface.

By default all windows are active (ON), but you can change this by setting the activity of specific windows ON or OFF.

Arguments

- **window_id** (integer)

Window number or ALL. Specifies the window(s) to have their status set

- **active_flag** (integer)

OFF or ON. OFF makes the selected window(s) inactive, ON makes window(s) active

Returns

boolean

Return type

Boolean

Example

```
// Turn off the activity flag for window #1
SetWindowActive(1, OFF);
// Make all current windows active
SetWindowActive(ALL, ON);
```

SetWindowFrame(window_id*[integer]*, frame_number*[integer]*) [static]

Description

Sets the current "frame" in the window(s) specified to frame_number.

The effect is immediate and the window(s) will be redrawn if necessary to show the requested frame

See the notes in GetWindowMaxFrame() on how frame number relates to state number

Arguments

- **window_id** (integer)

Window number or ALL

- **frame_number** (integer)

The frame number to set. Should be a +ve integer value in the range 1 to max #frames in window. Values greater than max #frames are truncated to this

Returns

boolean

Return type

Boolean

Example

```
// Set window #1 to display frame #10
SetWindowFrame(1, 10);
// Set all windows to display frame #3
SetWindowFrame(ALL, 3);
```

Examples

The following simple examples show how the functions above might be used. Further example scripts may be found in directory **$OASYS/d3plot_library/examples**

Capturing a sequence of static images to JPEG files

The following example loops over all frames in window #1 issuing the dialogue command "**/IMAGE JPG example_file_** *class="courier">nnn*.**jpg**" which will capture each frame in a separate JPEG file.

```
n = GetWindowMaxFrame(1); // Here window #1 is assumed to be current
for (i=1; i<=n; i++)
{
  SetWindowFrame(1, i);
  DialogueInput("/IMAGE JPG example_file_" + i + ".jpg");
}
```

Looping through states extracting cut-section results

The following example works through each state in the current model in turn, setting up a cut-section at the constant X coordinate of node #100, then rotates this section through 360 degrees in 5 degree increments, printing out the resulting forces and moments at each increment.

```
/* We need an array with 9 subscripts to hold data */
var data = new Array(9);
/* Turn on cut section display in window #1 */
SetCutSection(1, STATUS, ON);
/* Loop over all states in the model in turn */
n = GetNumberOf(STATE);
for(i=1; i<=n; i++)
{
    time = GetTime(i);
    Print("Time = " + time + "\n");
/* Set sections at const X at current position of node 100, and "get" the result
*/
    SetCutSection(1, CONST_X, 100);
/* Note that GetCutSection() gets the forces in the current state shown in the
window, and
** that to get results at a different time the state argument must be specified
as here. */
    info = GetCutSection(1, 0, 0, i);
/* The following uses origin and vectors mode to sweep the section through 360
degrees in
** 5 degree increments, extracting the forces and moments at each position */
    data[0] = info.origin[X]; /* Origin stays where it currently is */
    data[1] = info.origin[Y];
    data[2] = info.origin[Z];
    for(i=0; i<=360; i+=5)
    {
        st = Math.sin(i*0.017453);
        ct = Math.cos(i*0.017453);

        data[3] = ct; /* X axis vector */
        data[4] = 0.0;
        data[5] = st;
        data[6] = 0.0; /* XY plane vector */
        data[7] = st;
        data[8] = ct;
        SetCutSection(1, OR_AND_V, data);
        c = GetCutForces(1);
        Print("Forces = " + c.force[X] + ", " + c.force[Y] + ", " + c.force[Z] +
"\n");
        Print("Moments = " + c.moment[X] + ", " + c.moment[Y] + ", " +
c.moment[Z] + "\n");
        Print("Centroid = " + c.centroid[X] + ", " + c.centroid[Y] + ", " +
c.centroid[Z] + "\n");
        Print("Area = " + c.area + "\n\n");
```

```
    }
}
```

Calculating the max value and storing it as a UBIN component.

This example loops over all states in model #1 finding the maximum Sxx value of each solid and shell in the model, and storing it as a new scalar UBIN component in state #1.

```
/* If you have > 1 model then set the one you want. Model #1 is assumed in
** this example. */
SetCurrentModel(1);
/* It is assumed that only a single scalar value is required, and it is for
** solids & shells. So find out the number of states, solids and shells */
nstate = GetNumberOf(STATE);
nshell = GetNumberOf(SHELL);
nsolid = GetNumberOf(SOLID);
shell_env = new Array();
solid_env = new Array();
/* Create arrays to hold the max/min data. Here they are initialised to
** zero, which might not be a good choice if you are looking for max values
** and incoming results could be negative. Fill in your own initial value. */
for(j=1; j<=nshell; j++) shell_env[j] = 0.0;
for(j=1; j<=nsolid; j++) solid_env[j] = 0.0;

/* Loop over states collecting max data.
  **
  ** Note that making the outer loop the state is more efficient than making it
the
  ** element, since changing state is a more costly operation. */
for(i=1; i<=nstate; i++)
{
     SetCurrentState(i);
     Print("Doing state " + i + "\n");
     for(j=1; j<=nshell; j++)
     {
         c = GetData(SXX, SHELL, j, 1);
         if(c > shell_env[j]) shell_env[j] = c;
     }
     for(j=1; j<=nsolid; j++)
     {
        c = GetData(SXX, SOLID, j, 1);
        if(c > solid_env[j]) solid_env[j] = c;
     }
}
/* Now create a scalar user-defined binary component to hold the result */
icomp = CreateUbinComponent("Maximum of SXX", U_SOSH, U_SCALAR, REPLACE);
/* Data has to be stored at a state, so choose state 1 */
SetCurrentState(1);
/* Then populate this user-defined component, intg point #1. */
for(j=1; j<=nshell; j++) PutUbinData(icomp, SHELL, j, 1, shell_env[j]);
for(j=1; j<=nsolid; j++) PutUbinData(icomp, SOLID, j, 1, solid_env[j]);
```

Extracting data by ply from a composite analysis.

Post-processing composite analyses can be difficult because the elements have many integration points, and a physical ply may not use the same integration point in two adjacent elements, meaning that post-processing by "layer" (ie by integration point) is not helpful.

This script assumes that ply information has been created using a *PART_COMPOSITE card, with each ply assigned to a separate material id of the relevant ply number, and it sorts results into a separate UBIN component by ply. In this way a given UBIN component will show results for a single ply across the whole model.

```
// JavaScript to plot composite data for selected elements
// Date: 26 June 2008
// Version 1.0
n = get_window_max_frame(1);
set_current_model(1);
```

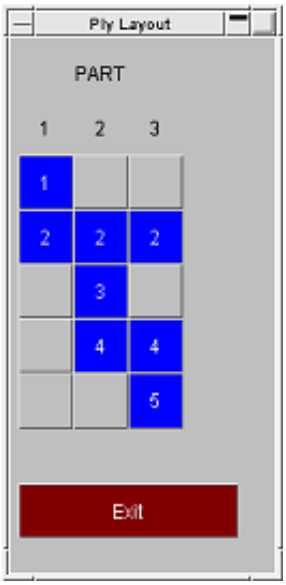
```
var a = GetWindowFrame(1);
SetCurrentState(a);
nsh = GetNumberOf(SHELL);
nip = GetNumberOf(NIP_S);
npa = GetNumberOf(PART);
var part = new Array();
var part_id = new Array();
var part_flag = 0;
var co1 = 0; // part counter
var co2 = 0; // ply counter
var ply = new Array();
var ply_id = new Array();
var ply_flag = 0;
var tmp1 = 0;
var offset = {}; // part, ip
var order1= {}; // part, ip
var order2= {}; // part, ip
var max1 = {};
var max2 = {};
var iter= 0; // iteration counter
var i_flag = 0;
var t_flag = 0;
// setup part array
for (i=1; i<=npa; i++) //shell
{
  part_flag = 0;
  a = GetLabel(PART, i);
  b = GetMid(PART, i, 1);
  if (b == 0) // check to see if the part is not a Part Composite
  {
    part_flag = 1;
  }
  for(j=1; j<=co1; j++) //part
  {
    if (a == part_id[j]) // check to see if the part is already defined
    {
      part_flag = 1;
    }
  }
  if (part_flag == 0)
  {
    co1 = co1 + 1;
    part[co1] = i;
    part_id[co1] = a;
  }
}
// setup ply order array
for (i=1; i<=co1; i++) // part
{
  order1[i] = new Object;
  for(j=1; j<=nip; j++) // ip
  {
    b = GetMid(PART, part[i], j);
    order1[i][j] = b;
  }
}
// setup ply id array
for (i=1; i<=co1; i++) //part
{
  for(j=1; j<=nip; j++) //ip
  {
    b = order1[i][j];
    if (b == 0) // ply doesn't exist
    {
      ply_flag = 1;
    }
    for (k=1; k<=co2; k++) //ply already created
    {
      if (b == ply_id[k])
      {
        ply_flag = 1;
```

```
      }
    }
    if (ply_flag == 0)
    {
      co2 = co2 + 1;
      ply_id[co2] = b;
      Message(co2 +" "+ b);
    }
    ply_flag = 0;
  }
}
//////////////////////////// proccessing loop
while (i_flag == 0)
{
  iter = iter + 1;
  Message("Iteration "+ iter);
  if(iter > 1)
  {
    for (i=1; i<=co1; i++) // part
    {
      for(j=1; j<=co2; j++) // ip
      {
        order1[i][j] = order2[i][j];
      }
    }
  }
//// ply order array
  for (i=1; i<=co2; i++) //ply
  {
    ply[i] = new object;
    for(j=1; j<=co1; j++) //part
    {
      ply[i][j] = new object;
      for(k=1; k<=co2; k++) //ply
      {
        b = order1[j][k];
        if (b == ply_id[i])
        {
          ply[i][j] = k
        }
      }
    }
  }
//// find max position for each ply
  for (i=1; i<=co2; i++)
  {
    max1[i] = 0;
    for(j=1; j<=co1; j++)
    {
      if (typeof(ply[i][j]) == "object")
      {
        ply[i][j] = 0;
      }

      max1[i] = Math.max(max1[i], ply[i][j]);
    }
  }
  for (i=1; i<=co2; i++)
  {
    Message(i +" "+ ply_id[i] + " "+ max1[i]);
  }
  // offset ply
  for (i=1; i<=co1; i++) // part
  {
    order2[i] = new Object;
    max = 0;
    pos2 = 0;
    for(j=1; j<=co2; j++) // ip
    {
      pos1 = j;
      b = order1[i][j]; // ply id
```

```
      for(k=1; k<=co2; k++) // ply
      {
        if(ply_id[k] == b)
        {
          max = max1[k]
        }
      }
      if(b != null && b != 0)
      {
        p_off = Math.max((pos2 - pos1 + 1),0);
        pos2 = Math.max(max, (p_off + pos1));
        order2[i][pos2] = b;
      }
    }
  }
  i_flag = 1;
  for (i=1; i<=co1; i++) // part
  {
    for(j=1; j<=co2; j++) // ip
    {
      if(order1[i][j] != order2[i][j]) i_flag = 0;
    }
  }
}
////////////////// create table
var we = new Window("Ply Layout", 0.2, 0.3, 0.5, 0.6 );
var x1 = 0;
var x2 = 0;
var y1 = 0
var y2 = 0;
var part = new Widget(we, Widget.LABEL, 0, Math.max((co1*10)+30, 70), 10, 20,
"PART");
part.justify=Widget.CENTRE;
for (i=1; i<=co2; i++) // ply
{
  t_flag = 1;
  y1 = (i*10) + 20;
  y2 = y1 + 10;
  for(j=1; j<=co1; j++) // part
  {
    x1 = (j*10) + 10;
    x2 = x1 + 10;
    if(i == 1)
    {
      var partl = new Widget(we, Widget.LABEL, x1, x2, 20, 30, ""+part_id[j]);
    }

    var mark = new Widget(we,
    Widget.BUTTON, x1, x2, y1, y2, " ");
    if (order2[j][i] != 0 && order2[j][i] != null)
    {
      mark.text= ""+order2[j][i];
      mark.background=Widget.BLUE;
      mark.foreground=Widget.WHITE;
      t_flag = 0;
    }
  }
  if(t_flag == 1)
  {
    var yn = i;
    i = co2+1;
  }
}
var ply = new Widget(we, Widget.LABEL, 10, 20, (yn*5)+25, (yn*5)+35, "PLY");
var exit = new Widget(we, Widget.BUTTON, 20, 60, (y1+20), (y2+20), "Exit");
exit.background = Widget.DARKRED;
exit.foreground = Widget.WHITE;
exit.onClick = ex_clicked;
we.Show(false)
////////////////////////////////////////////////////////////////////////
function ex_clicked()
```

```
{
  Exit();
}
////////////////////////////////////////////////////////////////////////////
```

Using the Window and Widget classes to present ply data graphically



This example demonstrates the use of Menu Window and Widget classes of the common API shared with PRIMER to present information to the user graphically, and to interact with them.

It goes through a laminate model using *PART_COMPOSITE and sorts the plies into usage by part. This information is then presented to the user as shown in the figure here and the script pauses until the user clicks on "Exit" in order to continue.

```
SetCurrentModel(1);
SetCurrentState(1);
nsh = GetNumberOf(SHELL);
nip = GetNumberOf(NIP_S);
npa = GetNumberOf(PART);
var part = new Array();
var part_id = new Array();
var part_flag = 0;
var co1 = 0; // part counter
var co2 = 0; // ply counter
var ply = new Array();
var ply_id = new Array();
var ply_flag = 0;
var tmp1 = 0;
var offset = {}; // part, ip
var order1= {}; // part, ip
var order2= {}; // part, ip
var max1 = {};
var max2 = {};
var iter= 0; // iteration counter
var i_flag = 0;
var t_flag = 0;
// setup part array
for (i=1; i<=nsh; i++) //shell
{
  part_flag = 0;
  a = GetPid(SHELL, i);
  a = GetLabel(PART, a);
  b = GetMid(SHELL, i, 1);
  for(j=1; j<=co1; j++) //part
  {
    if (a == part_id[j] || b == 0) // check to see if the part is already
defined or not a Part Composite
    {
      part_flag = 1;
    }
  }
  if (part_flag == 0)
  {
    co1 = co1 + 1;
    part[co1] = i;
```

```
    part_id[co1] = a;
  }
}
    // setup ply order array

for (i=1; i<=co1; i++) // part
{
  order1[i] = new Object;
  for(j=1; j<=nip; j++) // ip
  {
    b = GetMid(SHELL, part[i], j);
    order1[i][j] = b;
  }
}
// setup ply id array
for (i=1; i<=co1; i++) //part
{
  for(j=1; j<=nip; j++) //ip
  {
    b = order1[i][j];
    for (k=1; k<=co2; k++) //ply
    {
      if (b == ply_id[k] || b == 0)
      {
        ply_flag = 1
      }
    }
    if (ply_flag == 0)
    {
      co2 = co2 + 1;
      ply_id[co2] = b;
    }
    ply_flag = 0;
  }
}
/////////////////////////// processing loop
while (i_flag == 0)
{
  iter = iter + 1;
  Message("Iteration "+ iter);
  if(iter > 1)
  {
    for (i=1; i<=co1; i++) // part
    {
      for(j=1; j<=co2; j++) // ip
      {
        order1[i][j] = order2[i][j];
      }
    }
  }
//// ply order array
  for (i=1; i<=co2; i++) //ply
  {
    ply[i] = new Object;
    for(j=1; j<=co1; j++) //part
    {
      ply[i][j] = new Object;
      for(k=1; k<=co2; k++) //ply
      {
        b = order1[j][k];
        if (b == ply_id[i])
        {
          ply[i][j] = k
        }
      }
    }
  }
//// find max position for each ply
  for (i=1; i<=co2; i++)
  {
    max1[i] = 0;
```

```
    for(j=1; j<=co1; j++)
    {
      if (typeof(ply[i][j]) == "object")
      {
        ply[i][j] = 0;
      }
      max1[i] = Math.max(max1[i], ply[i][j]);
    }
  }
// offset ply
  for (i=1; i<=co1; i++) // part
  {
    tmp1 = 0;
    offset[i] = new Object;
    for(j=1; j<=co2; j++) // ip
    {
      b = order1[i][j];
      for(k=1; k<=co2; k++) // ply
      {
        if(ply_id[k] == b)
        {
          if(max1[k] > j)
          {
            tmp1 = max1[k] - j;
          }
        }
      }
      offset[i][j] =tmp1
    }
  }
  for (i=1; i<=co1; i++) // part
  {
    order2[i] = new Object;
    for(j=1; j<=co2; j++) // ip
    {
      a = offset[i][j];
      b = order1[i][j];
      order2[i][j+a] = b;
    }
  }
  i_flag = 1;
  for (i=1; i<=co1; i++) // part
  {
    for(j=1; j<=co2; j++) // ip
    {
      if(offset[i][j] != 0) i_flag = 0;
    }
  }
}
////////////////// create table

var we = new Window("Ply Layout", 0.2, 0.3, 0.5, 0.6 );
var x1 = 0;
var x2 = 0;
var y1 = 0
var y2 = 0;
var part = new Widget(we, Widget.LABEL, (co1*5)+15, (co1*5)+25, 10, 20, "PART");
part.justify=Widget.CENTRE;
for (i=1; i<=co2; i++) // ply
{
  t_flag = 1;
  y1 = (i*10) + 20;
  y2 = y1 + 10;
  for(j=1; j<=co1; j++) // part
  {
    x1 = (j*10) + 10;
    x2 = x1 + 10;
    if(i == 1)
    {
      Message("a"+ part_id[j]);
      var partl = new Widget(we, Widget.LABEL, x1, x2, 20, 30, ""+part_id[j]);
```

```
    }
    var mark = new Widget(we, Widget.BUTTON, x1, x2, y1, y2, " ");
    if (order2[j][i] != 0 && order2[j][i] != null)
    {
      mark.text= ""+order2[j][i];
      mark.background=Widget.BLUE;
      mark.foreground=Widget.WHITE;
      t_flag = 0;
    }
  }
  if(t_flag == 1)
  {
    var yn = i;
    i = co2+1;
  }
}
var ply = new Widget(we, Widget.LABEL, 10, 20, (yn*5)+25, (yn*5)+35, "PLY");
var exit = new Widget(we, Widget.BUTTON, 20, 60, (y1+20), (y2+20), "Exit");
exit.background = Widget.DARKRED;
exit.foreground = Widget.WHITE;
exit.onClick = ex_clicked;
we.Show(false);
////////////////////////////////////////////////////////////////////////
function ex_clicked()
{
  Exit();
}
```

Using the File class to write data to file.

This example extracts maximum and minimum Z displacement values of nodes and writes them to a file "**reporter_variables**" that can be used to set up variables in Reporter. It demonstrates the use of the File class to open a file, write to it, and close it.

```
var fence_part_min = 4;
var fence_part_max = 7;
var max_states;
var istate;
var ipart;
var nnode;
var inode;
var objElem;
var internal_pid;
var external_pid;
var z_max = 0;
var z_min = 0;
var max_node = 0;
var min_node = 0;
var max_state = 0;
var min_state = 0;
var f;
/* Get the number of states in the current model */
max_states = GetNumberOf(STATE);
/* Get the number of nodes in the current model */
nnode = GetNumberOf(NODE);
print("Start of loop\n");
/* Loop over each state */
for(istate=1; istate<=max_states; istate++)
{
  SetCurrentState(istate);
  print("State " + istate + "\n");
/* Loop over each node */
  for(inode=1; inode<=nnode; inode++)
  {
/* Get shell elements at node */
    if(objElem = GetElemsAtNode(inode, SHELL))
    {
      if(objElem.nn > 0)
      {
```

```
/* Get the external PID */

        internal_pid = GetPid(SHELL, objElem.list[0]);
        external_pid = GetLabel(PART, internal_pid);
/* Check it against parts to test */

        if(external_pid >= fence_part_min && external_pid<=fence_part_max)
        {
          temp = GetData(DZ, NODE, inode);
          if(temp > z_max)
          {
            max_state = istate; /* Store the state the maximum occurs */
            max_node = inode; /* Store the node the maximum occurs at */
            z_max = temp; /* Store the maximum */
          }
          if(temp < z_min)
          {
            min_state = istate; /* Store the state the minimum occurs */
            min_node = inode; /* Store the node the minimum occurs at */
            z_min = temp; /* Store the minimum */
          }
        }
      }
    }
  }
}
/* Open a file to write the variables to */
f = new File("./reporter_variables", File.WRITE);
/* Write to the file */
f.Writeln("VAR Z_MAX DESCRIPTION='Maximum z displacement' VALUE='" + z_max +
"'");
f.Writeln("VAR Z_MIN DESCRIPTION='Minimum z displacement' VALUE='" + z_min +
"'");
f.Writeln("VAR Z_MAX_NODE DESCRIPTION='Node with maximum z displacement'
VALUE='" + max_node + "'");
f.Writeln("VAR Z_MIN_NODE DESCRIPTION='Node with minimum z displacement'
VALUE='" + min_node + "'");
f.Writeln("VAR Z_MAX_STATE DESCRIPTION='State with maximum z displacement'
VALUE='" + max_state + "'");
f.Writeln("VAR Z_MIN_STATE DESCRIPTION='State with minimum z displacement'
VALUE='" + min_state + "'");
/* Close the file */
f.Close();
```

Next section

global class

The global class is the main JavaScript class. More...

The T/HIS JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- AllocateFlag()
- ClearFlag(flag*[Flag]*)
- DialogueInput(command*[string]*)
- DialogueInputNoEcho(command*[string]*)
- DisableGraphWindowUpdates()
- DisableMenuUpdates()
- EnableGraphWindowUpdates()
- EnableMenuUpdates()
- ErrorMessage(string*[Any valid javascript type]*)
- Execute(data*[object]*)
- Exit(write hook interrupt (optional)*[boolean]*)
- GetCurrentDirectory()
- GetFtcfVar(name*[string]*)
- GetInstallDirectory()
- GetPreferenceValue(program*[string]*, name*[string]*)
- GetStartInDirectory()
- Getenv(name*[string]*)
- Message(string*[Any valid javascript type]*)
- MilliSleep(time*[integer]*)
- NumberToString(number*[integer/real]*, width*[integer]*, pref_int (optional)*[boolean]*)
- OpenManual(program*[string]*, page*[string]*)
- Plot()
- Print(string*[Any valid javascript type]*)
- Println(string*[Any valid javascript type]*)
- ReturnFlag(flag*[Flag]*)
- SetCurrentDirectory(directory path*[string]*)
- SetFtcfVar(name*[string]*)
- Sleep(time*[integer]*)
- System(string*[Any valid javascript type]*)
- THISLinkStatus()
- Unix()
- UpdateCurveMenu()
- WarningMessage(string*[Any valid javascript type]*)
- Windows()

Detailed Description

The global class declares the global object in JavaScript that contains the global properties and methods. As well as the core JavaScript methods, T/HIS also defines other additional ones. e.g. Message(), Print() etc. See the documentation below for more details.

Details of functions

AllocateFlag() [static]

Description

Allocate a flag for use in the script. See also ReturnFlag() and Once allocated the flag is automatically cleared for all entity types and all the curves currently in T/HIS.

Arguments

No arguments

Returns

Flag (integer)

Return type

Number

Example

To allocate a flag

```
var flag = AllocateFlag();
```

ClearFlag(flag*[Flag]*) [static]

Description

Clears a flag on all curves and entity types.

Arguments

- **flag** (Flag)

The flag to return.

Returns

No return value.

Example

To clear flag f:

```
ClearFlag(f);
```

DialogueInput(command*[string]*) [static]

Description

Execute one or more lines of command line dialogue input.

Arguments

- **command** (string)

Command to execute (as if it had been typed into the dialogue box)

This argument can be repeated if required
Alternatively a single array argument containing the multiple values can be given

Returns

No return value

Example

To mulitply curves 1 and 2 by 10:

```
DialogueInput("/op mul #1 10 #", "/op mul #2 10 #");
```

Note that each call to DialogueInput starts afresh at the top of the T/HIS command line "tree", so where multiple commands need to be given at sub-menu levels they need to be included in a single call.

DialogueInputNoEcho(command*[string]*) [static]

Description

Execute one or more lines of command line dialogue input **with no echo of commands to dialogue box**.

Arguments

- **command** (string)

Command to execute (as if it had been typed into the dialogue box)

This argument can be repeated if required
Alternatively a single array argument containing the multiple values can be given

Returns

No return value

Example

To mulitply curves 1 and 2 by 10:

```
DialogueInputNoEcho("/op mul #1 10 #", "/op mul #2 10 #");
```

As with DialogueInput above each call starts at the top of the T/HIS command tree structure, so any commands destined for sub-menus must all be arguments to a single call.

DisableGraphWindowUpdates() [static]

Description

Disable Graph Window updates.

Arguments

No arguments

Returns

No return value

Example

Turn off graph window updates

```
DisableGraphWindowUpdates();
```

DisableMenuUpdates() [static]

Description

Disable menu system updates.

Arguments

No arguments

Returns

No return value

Example

Disable menu system updates

```
DisableMenuUpdates();
```

EnableGraphWindowUpdates() [static]

Description

Enable Graph Window updates.

Arguments

No arguments

Returns

No return value

Example

Turn off graph window updates

```
EnableGraphWindowUpdates()
```

EnableMenuUpdates() [static]

Description

Enable menu system updates.

Arguments

No arguments

Returns

No return value

Example

Enable menu system updates

```
EnableMenuUpdates();
```

ErrorMessage(string*[Any valid javascript type]*) [static]

Description

Print an error message to the dialogue box **adding a carriage return**.

Arguments

- **string** (Any valid javascript type)

The string/item that you want to print

Returns

No return value

Example

To print the title of model object m as an error to the dialogue box

```
ErrorMessage("The title is " + m.title);
```

Execute(data*[object]*) [static]

Description

Execute a program or script outside T/HIS and get the standard output and error streams.

Arguments

- **data** (object)

Execute data

Object has the following properties:

Name	Type	Description
arguments (optional)	Array of strings	The arguments to pass to program
program	string	The program you want to run. Note that on Linux this will consider PATH when resolving executable filenames without an absolute path. If you want to run something from the current directory and you do not have '.' in your PATH then you will need to write './something' as the program.

Returns

Object with the following properties:

Name	Type	Description
status	integer	The exit code from the program/script
stderr	string	The standard error output from the program/script
stdout	string	The standard output from the program/script

Return type

object

Example

To run script "example.bat" with arguments "foo" and "bar":

```
var output = Execute( { program: 'example.bat', arguments: [ 'foo', 'bar' ]} );
var text   = output.stdout;
var errors = output.stderr;
var ecode  = output.status;
```

Exit(write hook interrupt (optional)*[boolean]*) [static]

Description

Exit script

Arguments

- **write hook interrupt (optional)** (boolean)

If Exit() is called from a write_hook.js script, the first argument will be processed as in the following: If the argument is provided and set to "true", it is used to interrupt the write out of the model, so that the script exits without anything being written out. An argument value of "false" exits the script and allows the model to be written out as normal. An example of this function's use in a Write Hook script can be found at $OA_INSTALL/primer_library/scripts/hooks/example_write_hook.js.

Returns

No return value

Example

Exit with

```
Exit();
```

GetCurrentDirectory() [static]

Description

Get the current working directory

Arguments

No arguments

Returns

String containing current working directory

Return type

String

Example

To get the current directory:

```
var cwd = GetCurrentDirectory();
```

GetFtcfVar(name*[string]*) [static]

Description

Get the value of a FAST-TCF variable

Arguments

- **name** (string)

The FAST-TCF variable name (case independent)

Returns

String containing variable value or null if variable does not exist

Return type

String

Example

To get the value for FAST-TCF variable Job

```
var job_name = GetFtcfVar("Job");
```

GetInstallDirectory() [static]

Description

Get the directory in which executables are installed. This is the OA_INSTALL environment variable, or if that is not set the directory in which the current executable is installed. Returns NULL if not found

Arguments

No arguments

Returns

string

Return type

String

Example

To get the install directory:

```
var install_dir = GetInstallDirectory();
```

GetPreferenceValue(program*[string]*, name*[string]*) [static]

Description

Get the Preference value with the given string in the any of admin ("OA_ADMIN") or install ("OA_INSTALL") or home ("OA_HOME") directory oa_pref

Arguments

- **program** (string)

The program name string : Valid values are ’All’, ’D3PLOT’, ’PRIMER’, ’REPORTER’, ’SHELL’, ’T/HIS’

- **name** (string)

The preference name string

Returns

: String containing preference value or null if preference string is not present in any oa_pref. Also if none of the above environment variables are not present, then API simply returns null. While returning preference value, locked preference value in admin and then install oa_pref takes precedence over home oa_pref. If preference is not locked in any of these oa_pref, preference in home directory oa_pref is returned.

Return type

String

Example

To get the preference value:

```
var pref_list = GetPreferenceValue(’All’, "font_size");
```

GetStartInDirectory() [static]

Description

Get the directory passed to T/HIS by the -start_in command line argument

Arguments

No arguments

Returns

String containing start_in directory or NULL if not set

Return type

String

Example

To get the start_in directory:

```
var start_in = GetStartInDirectory();
```

Getenv(name*[string]*) [static]

Description

Get the value of an environment variable

Arguments

- **name** (string)

The environment variable name

Returns

String containing variable value or null if variable does not exist

Return type

String

Example

To get the value for environment variable HOME

```
var home = Getenv("HOME");
```

Message(string*[Any valid javascript type]*) [static]

Description

Print a message to the dialogue box **adding a carriage return**.

Arguments

- **string** (Any valid javascript type)

The string/item that you want to print. If '\r' is added to the end of the string then instead of automatically adding a carriage return in the dialogue box, the next message will overwrite the current one. This may be useful for giving feedback to the dialogue box when doing an operation.

Returns

No return value

Example

To print the title of model object m as a message to the dialogue box

```
Message("The title is " + m.title);
```

MilliSleep(time*[integer]*) [static]

Description

Pause execution of the script for *time* milliseconds. See also Sleep()

Arguments

- **time** (integer)

Number of milliseconds to pause for

Returns

No return value

Example

To pause for 500 milliseconds

```
MilliSleep(500);
```

NumberToString(number*[integer/real]*, width*[integer]*, pref_int (optional)*[boolean]*) [static]

Description

Formats a number to a string with the specified width.

Arguments

- **number** (integer/real)

The number you want to format.

- **width** (integer)

The width of the string you want to format it to (must be less than 80).

- **pref_int (optional)** (boolean)

By default only integer values inside the single precision 32 bit signed integer limit of approximately +/-2e9 are formatted as integers, all other numeric values are formatted as floats. With this argument set to TRUE then integer values up to the mantissa precision of a 64 bit float, approximately +/-9e15, will also be formatted as integers.

Returns

String containing the number

Return type

String

Example

To write the number 1.2345e+6 to a string 10 characters wide

```
var str = NumberToString(1.2345e+6, 10);
```

OpenManual(program*[string]*, page*[string]*) [static]

Description

Open the Oasys manuals at a requested page

Arguments

- **program** (string)

The program manual to open. Can be "primer", "d3plot" or "this"

- **page** (string)

The page to open in the manual, e.g. "running-this.html"

Returns

true if successful, false if not

Return type

Boolean

Example

To open the T/HIS manual on the running-this.html page

```
OpenManual("this", "running-this.html");
```

Plot() [static]

Description

Updates all the T/HIS graphs.

Arguments

No arguments

Returns

No return value

Example

Update all graphs

```
Plot();
```

Print(string*[Any valid javascript type]*) [static]

Description

Print a string to stdout. **Note that a carriage return is not added**.

Arguments

- **string** (Any valid javascript type)

The string/item that you want to print

Returns

No return value

Example

To print string "Hello, world!"

```
Print("Hello, world!");
```

To print the title of model object m with a carriage return

```
print("The title is " + m.title + "\n");
```

Println(string*[Any valid javascript type]*) [static]

Description

Print a string to stdout **adding a carriage return**.

Arguments

- **string** (Any valid javascript type)

The string/item that you want to print

Returns

No return value

Example

To print string "Hello, world!" automatically adding a carriage return

```
Println("Hello, world!");
```

To print the title of model object m, automatically adding a carriage return

```
Println("The title is " + m.title);
```

ReturnFlag(flag*[Flag]*) [static]

Description

Return a flag used in the script. See also AllocateFlag() and

Arguments

- **flag** (Flag)

The flag to return.

Returns

No return value.

Example

To return flag f:

```
ReturnFlag(f);
```

SetCurrentDirectory(directory path*[string]*) [static]

Description

Sets the current working directory.

Arguments

- **directory path** (string)

Path to the directory you would like to change into.

Returns

true if successful, false if not

Return type

Boolean

Example

To change into the directory "/data/test" exists

```
SetCurrentDirectory("/data/test")
```

SetFtcfVar(name*[string]*) [static]

Description

Set the value of a FAST-TCF variable. If the variable already exists then it's value is updated

Arguments

- **name** (string)

The FAST-TCF variable name (case independent)

Returns

String containing variable value or null if variable does not exist

Return type

String

Example

To create a new FAST-TCF variable called run_number with the value "10"

```
var home = SetFtcfVar("run_number","10");
```

Sleep(time*[integer]*) [static]

Description

Pause execution of the script for *time* seconds. See also MilliSleep()

Arguments

- **time** (integer)

Number of seconds to pause for

Returns

No return value

Example

To pause for 2 seconds

```
Sleep(2);
```

System(string*[Any valid javascript type]*) [static]

Description

Do a system command outside T/HIS. To run an external command and get the output then please use Execute() instead.

Arguments

- **string** (Any valid javascript type)

The system command that you want to do

Returns

integer (probably zero if command successful but is implementation-dependant)

Return type

Number

Example

To make the directory "example"

```
System("mkdir example");
```

THISLinkStatus() [static]

Description

Get the status of T/HIS in the D3PLOT link

Arguments

No arguments

Returns

Integer specifying status of T/HIS (Constant.STANDALONE, Constant.DOCKED, Constant.MENUS_UNDOCKED, Constant.ALL_UNDOCKED)

Return type

Number

Example

To get the current status of T/HIS:

```
var status = THISLinkStatus();
```

Unix() [static]

Description

Test whether script is running on a Unix/Linux operating system. See also Windows()

Arguments

No arguments

Returns

true if Unix/Linux, false if not

Return type

Boolean

Example

To test if the OS is Unix

```
if ( Unix() )
```

UpdateCurveMenu() [static]

Description

Updates the scroll bar in the Curve Manager. Useful if your script has created lots of new curves and you want to update the menu (otherwise, the scroll bar range will only be updated when the script exits).

Arguments

No arguments

Returns

No return value

Example

Update curve slider

```
UpdateCurveMenu();
```

WarningMessage(string*[Any valid javascript type]*) [static]

Description

Print a warning message to the dialogue box **adding a carriage return**.

Arguments

- **string** (Any valid javascript type)

The string/item that you want to print

Returns

No return value

Example

To print the title of model object m as a warning to the dialogue box

```
WarningMessage("The title is " + m.title);
```

Windows() [static]

Description

Test whether script is running on a Windows operating system. See also Unix()

Arguments

No arguments

Returns

true if Windows, false if not

Return type

Boolean

Example

To test if the OS is Windows

```
if ( Windows() )
```

Colour class

The Colour class contains constants relating to colours. More...

The T/HIS JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- GetFromName(name*[string]*)
- RGB(red*[integer]*, green*[integer]*, blue*[integer]*)

Colour constants

Name	Description
Colour.BACKGROUND	Background colour
Colour.BLACK	Colour black
Colour.BLUE	Colour blue
Colour.CYAN	Colour cyan
Colour.DARK_GREEN	Colour dark green
Colour.DARK_GREY	Colour dark grey
Colour.DARK_MAGENTA	Colour dark magenta
Colour.FOREGROUND	Foreground colour
Colour.GOLD	Colour gold
Colour.GREEN	Colour green
Colour.HOT_PINK	Colour hot pink
Colour.INDIGO	Colour indigo
Colour.LIGHT_GREY	Colour light grey
Colour.LIGHT_PINK	Colour light pink
Colour.LIME	Colour lime
Colour.MAGENTA	Colour magenta
Colour.MAROON	Colour maroon
Colour.MEDIUM_BLUE	Colour medium blue
Colour.MEDIUM_GREEN	Colour medium green
Colour.MEDIUM_GREY	Colour medium grey
Colour.NAVY	Colour navy
Colour.OLIVE	Colour olive
Colour.ORANGE	Colour orange
Colour.PALE_YELLOW	Colourpale yellow

Colour.PINK	Colour pink
Colour.PURPLE	Colour purple
Colour.RED	Colour red
Colour.SEA_GREEN	Colour sea green
Colour.SKY	Colour sky
Colour.TURQUOISE	Colour turquoise
Colour.USER_1	Colour user defined 1
Colour.USER_2	Colour user defined 2
Colour.USER_3	Colour user defined 3
Colour.USER_4	Colour user defined 4
Colour.USER_5	Colour user defined 5
Colour.USER_6	Colour user defined 6
Colour.USER_n (n = 1 to 150)	n-th user defined colour
Colour.WHITE	Colour white
Colour.YELLOW	Colour yellow

Detailed Description

The Colour class is used to define colours:

```
p.colour = Colour.RED;
```

Details of functions

GetFromName(name*[string]*) [static]

Description

Returns the colour for a given core or user colour name

Arguments

- **name** (string)

The name of the colour, for example red or user_green or green/cyan.

Returns

colour value (integer)

Return type

Number

RGB(red*[integer]*, green*[integer]*, blue*[integer]*) [static]

Description

Creates a colour from red, green and blue components

Arguments

- **red** (integer)

red component of colour (0-255).

- **green** (integer)

green component of colour (0-255).

- **blue** (integer)

blue component of colour (0-255).

Returns

colour value (integer)

Return type

Number

Component class

The following component constants can be used in GetDataFlagged() in T/HIS.

Component constants

Constants for MODEL

Name	Description
Component.GSTP	Time step
Component.GKE	Kinetic energy
Component.GIE	Internal energy
Component.GSWE	Stonewall energy
Component.GSPE	Spring and damper energy
Component.GHG	Hourglass energy
Component.GSDE	System damping energy
Component.GJE	Joint internal energy
Component.GSIE	Sliding interface energy
Component.GEW	External work
Component.GRBE	Rigid Body stopper energy
Component.GTE	Total energy
Component.GTER	Total/initial energy
Component.GVX	Average X velocity
Component.GVY	Average Y velocity
Component.GVZ	Average Z velocity
Component.GTZC	Time per zone cycle
Component.GMASS	Total mass
Component.GMADD	Added mass
Component.GPM	%age Mass increase
Component.GEKE	Eroded Kinetic energy
Component.GEIE	Eroded Internal energy
Component.GEHG	Eroded Hourglass energy
Component.GER	Energy Ratio w/o Eroded
Component.DRCE	Current Distortional KE
Component.DRMX	Maximum Distortional KE
Component.DRCO	Convergence Factor
Component.DRKE	Total Kinetic energy
Component.LKE	Lumped Kinetic energy
Component.GMPE	Mat Plastic energy
Component.GMEE	Mat Elastic energy

Component.GMDE	Mat Damage energy
Component.GDIE	Dissipated IE
Component.GDKE	Dissipated KE
Component.GDE	Drilling energy

Constants for PART

Name	**Description**
Component.GKE	Kinetic energy
Component.GIE	Internal energy
Component.GHG	Hourglass energy
Component.GTE	Total energy
Component.GMX	X momentum
Component.GMY	Y momentum
Component.GMZ	Z momentum
Component.GVX	Average X velocity
Component.GVY	Average Y velocity
Component.GVZ	Average Z velocity
Component.GMASS	Mass
Component.GAM	Added mass
Component.GEKE	Eroded Kinetic energy
Component.GEIE	Eroded Internal energy
Component.GMPE	Mat Plastic energy
Component.GMEE	Mat Elastic energy
Component.GMDE	Mat Damage energy

Constants for NODE

Name	**Description**
Component.TEMP	Temperature
Component.DX	X Displacement
Component.DY	Y Displacement
Component.DZ	Z Displacement
Component.DM	Displacement Magnitude
Component.VX	X Velocity
Component.VY	Y Velocity
Component.VZ	Z Velocity
Component.VM	Velocity Magnitude
Component.AX	X Acceleration
Component.AY	Y Acceleration

Component.AZ	Z Acceleration
Component.AM	Acceleration Magnitude
Component.CX	X co-ordinate
Component.CY	Y co-ordinate
Component.CZ	Z co-ordinate
Component.CV	Current Vector
Component.BX	Basic X co-ordinate
Component.BY	Basic Y co-ordinate
Component.BZ	Basic Z co-ordinate
Component.BV	Basic Vector
Component.RDX	X rotation
Component.RDY	Y rotation
Component.RDZ	Z rotation
Component.RDM	Rotation Magnitude
Component.RVX	X rotational velocity
Component.RVY	Y rotational velocity
Component.RVZ	Z rotational velocity
Component.RVM	Rotation Vel Magnitude
Component.RAX	X rotational acceleration
Component.RAY	Y rotational acceleration
Component.RAZ	Z rotational acceleration
Component.RAM	Rotation Accel Magnitude
Component.TFX	X Thermal Flux
Component.TFY	Y Thermal Flux
Component.TFZ	Z Thermal Flux
Component.TFM	Thermal Flux Magnitude
Component.TTOP	Top Temperature
Component.TBOT	Bottom Temperature

Constants for SOLID

Name	**Description**
Component.SXX	Stress in XX
Component.SYY	Stress in YY
Component.SZZ	Stress in ZZ
Component.SXY	Stress in XY
Component.SYZ	Stress in YZ
Component.SZX	Stress in ZX
Component.SMAX	MAX principal stress
Component.SMIN	MIN principal stress

Component.SMS	MAX shear stress
Component.SVON	von Mises stress
Component.SAV	Average stress (Pressure)
Component.STR	Triaxiality Factor
Component.EPL	Effective plastic strain
Component.EXX	Strain in XX
Component.EYY	Strain in YY
Component.EZZ	Strain in ZZ
Component.EXY	Strain in XY
Component.EYZ	Strain in YZ
Component.EZX	Strain in ZX
Component.EMAX	MAX principal strain
Component.EMIN	MIN principal strain
Component.EMS	MAX shear strain
Component.EVON	von Mises strain
Component.EAV	Average strain
Component.PEMAG	Plastic Strain Magnitude
Component.SOX	Extra data

Constants for BEAM

Name	**Description**
Component.BFX	Axial force
Component.BFY	Shear force in Y
Component.BFZ	Shear force in Z
Component.BMXX	Torsional moment
Component.BMYY	Moment in Y
Component.BMZZ	Moment in Z
Component.BSAX	Axial strain
Component.BPE1	Bending energy: end 1
Component.BPE2	Bending energy: end 2
Component.BRY1	Y rotation: end 1
Component.BRY2	Y rotation: end 2
Component.BRZ1	Z rotation: end 1
Component.BRZ2	Z rotation: end 2
Component.BRXX	Torsional rotation
Component.BMY1	Y Bending moment: end 1
Component.BMY2	Y Bending moment: end 2
Component.BMZ1	Z Bending moment: end 1
Component.BMZ2	Z Bending moment: end 2

Component.BACE	Axial collapse energy
Component.BIE	Internal energy
Component.BSXX	Axial stress
Component.BSXY	XY Shear stress
Component.BSZX	ZX Shear stress
Component.BEP	Effective plastic strain
Component.BEAX	Axial strain
Component.BDX	Relative Axial displacement
Component.BDY	Relative S - Displacement
Component.BDZ	Relative T - Displacement
Component.BRY	Rotation in S
Component.BRZ	Rotation in T
Component.BDNA	Relative Axial force
Component.BDNS	Resultant S - Force
Component.BDNT	Resultant T - Force
Component.BDMA	Axial moment
Component.BDMS	Moment in S
Component.BDMT	Moment in T
Component.BDDX	Axial Direction X
Component.BDDY	Axial Direction Y
Component.BDDZ	Axial Direction Z
Component.BDSX	S - Direction X
Component.BDSY	S - Direction Y
Component.BDSZ	S - Direction Z
Component.BDTX	T - Direction X
Component.BDTY	T - Direction Y
Component.BDTZ	T - Direction Z
Component.BEX	Extra data

Constants for SHELL

Name	**Description**
Component.SXX	Stress in XX
Component.SYY	Stress in YY
Component.SZZ	Stress in ZZ
Component.SXY	Stress in XY
Component.SYZ	Stress in YZ
Component.SZX	Stress in ZX
Component.SMAX	MAX principal stress
Component.SMIN	MIN principal stress

Component.SMS	MAX shear stress
Component.SVON	von Mises stress
Component.SAV	Average stress (Pressure)
Component.STR	Triaxiality Factor
Component.EPS	Effective plastic strain
Component.EXX	Strain in XX
Component.EYY	Strain in YY
Component.EZZ	Strain in ZZ
Component.EXY	Strain in XY
Component.EYZ	Strain in YZ
Component.EZX	Strain in ZX
Component.EMAX	MAX principal strain
Component.EMIN	MIN principal strain
Component.EMS	MAX shear strain
Component.EVON	von Mises strain
Component.EAV	Average strain
Component.PEMAG	Plastic Strain Magnitude
Component.RMX	Moment in X
Component.RMY	Moment in Y
Component.RMXY	Moment in XY
Component.RQX	Shear force in X
Component.RQY	Shear force in Y
Component.RFX	Normal force in X
Component.RFY	Normal force in Y
Component.RFXY	Normal force in XY
Component.THK	Thickness
Component.EDEN	Internal energy density
Component.SHX	Extra data

Constants for THICK_SHELL

Name	**Description**
Component.SXX	Stress in XX
Component.SYY	Stress in YY
Component.SZZ	Stress in ZZ
Component.SXY	Stress in XY
Component.SYZ	Stress in YZ
Component.SZX	Stress in ZX
Component.SMAX	MAX principal stress
Component.SMIN	MIN principal stress

Component.SMS	MAX shear stress
Component.SVON	von Mises stress
Component.SAV	Average stress (Pressure)
Component.STR	Triaxiality Factor
Component.EPL	Effective plastic strain
Component.EXX	Strain in XX
Component.EYY	Strain in YY
Component.EZZ	Strain in ZZ
Component.EXY	Strain in XY
Component.EYZ	Strain in YZ
Component.EZX	Strain in ZX
Component.EMAX	MAX principal strain
Component.EMIN	MIN principal strain
Component.EMS	MAX shear strain
Component.EVON	von Mises strain
Component.EAV	Average strain
Component.PEMAG	Plastic Strain Magnitude
Component.SHX	Extra data

Constants for RIGIDWALL

Name	**Description**
Component.FN	Normal force
Component.FX	Global X force
Component.FY	Global Y force
Component.FZ	Global Z force
Component.EN	Energy

Constants for SPRING

Name	**Description**
Component.SP_F	Resultant Force
Component.SP_E	Elongation
Component.SP_FE	Res Force v Elongation
Component.SP_FX	Global X force
Component.SP_FY	Global Y force
Component.SP_FZ	Global Z force
Component.SP_EN	Energy
Component.SP_M	Resultant Moment

Component.SP_R	Rotation
Component.SP_MR	Res Moment v Rotation
Component.SP_MX	Moment in X
Component.SP_MY	Moment in Y
Component.SP_MZ	Moment in Z

Constants for SEATBELT

Name	Description
Component.SB_F	Force
Component.SB_S	Strain
Component.SB_FS	Force v Strain
Component.SB_L	Current Length

Constants for RETRACTOR

Name	Description
Component.RT_F	Force
Component.RT_P	Pullout
Component.RT_FP	Force v Pullout

Constants for SLIPRING

Name	Description
Component.SR_P	Pull through
Component.SR_W	Warp Angle
Component.SR_S	Skew Angle
Component.SR_F	Friction Coeff
Component.SR_N	Normal Force
Component.SR_B1	Side 1 Belt Force
Component.SR_B2	Side 2 Belt Force

Constants for PRETENSIONER

Name	Description
Component.PR_FI	’Fired’ (= 1)

Constants for CONTACT

Name	Description

Component.CFXA	A Surface X force
Component.CFXS	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. A Surface X force (alternative name for Component.CFXA) [deprecated]
Component.CFYA	A Surface Y force
Component.CFYS	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. A Surface Y force (alternative name for Component.CFYA) [deprecated]
Component.CFZA	A Surface Z force
Component.CFZS	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. A Surface Z force (alternative name for Component.CFZA) [deprecated]
Component.CFMA	A Surface Force Mag
Component.CFMS	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. A Surface Force Mag (alternative name for Component.CFMA) [deprecated]
Component.CFXB	B Surface X force
Component.CFX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. B Surface X force (alternative name for Component.CFXB) [deprecated]
Component.CFYB	B Surface Y force
Component.CFY	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. B Surface Y force (alternative name for Component.CFYB) [deprecated]
Component.CFZB	B Surface Z force
Component.CFZ	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. B Surface Z force (alternative name for Component.CFZB) [deprecated]
Component.CFMB	B Surface Force Mag
Component.CFM	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. B Surface Force Mag (alternative name for Component.CFMB) [deprecated]
Component.CMXA	A Surface X moment
Component.CMXS	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. A Surface X moment (alternative name for Component.CMXA) [deprecated]
Component.CMYA	A Surface Y moment
Component.CMYS	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. A Surface Y moment (alternative name for Component.CMYA) [deprecated]
Component.CMZA	A Surface Z moment
Component.CMZS	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. A Surface Z moment (alternative name for Component.CMZA) [deprecated]
Component.CMXB	B Surface X moment
Component.CMX	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. B Surface X moment (alternative name for Component.CMXB) [deprecated]
Component.CMYB	B Surface Y moment

Component.CMY	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. B Surface Y moment (alternative name for Component.CMYB) [deprecated]
Component.CMZB	B Surface Z moment
Component.CMZ	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. B Surface Z moment (alternative name for Component.CMZB) [deprecated]
Component.CMA	A Surface Mass
Component.CMS	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. A Surface Mass (alternative name for Component.CMA) [deprecated]
Component.CMB	B Surface Mass
Component.CMM	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. B Surface Mass (alternative name for Component.CMB) [deprecated]
Component.CENA	A Surface side energy
Component.CENS	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. A Surface side energy (alternative name for Component.CENA) [deprecated]
Component.CENB	B Surface side energy
Component.CENM	This constant is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions. B Surface side energy (alternative name for Component.CENB) [deprecated]
Component.CFRI	Frictional energy
Component.CTEN	Total energy

Constants for NODE_GROUP

Name	Description
Component.RFX	X force
Component.RFY	Y force
Component.RFZ	Z force
Component.RFM	Force Magnitude
Component.EN	Energy
Component.LFX	Local X force
Component.LFY	Local Y force
Component.LFZ	Local Z force
Component.GRFX	X force
Component.GRFY	Y force
Component.GRFZ	Z force
Component.GRFM	Force Magnitude
Component.GEN	Energy

Constants for AIRBAG

Name	Description

Component.PR	Pressure
Component.EPR	Exact Pressure
Component.VOL	Volume
Component.DE	Density
Component.EDE	Exact Density
Component.IE	Internal energy
Component.EIE	Exact Internal energy
Component.IN	Mass flow rate in
Component.OU	Mass flow rate out
Component.MIN	Mass in
Component.MOU	Mass out
Component.MASS	Total mass
Component.SA	Surface area
Component.TEMP	Gas temperature
Component.FR	Reaction force
Component.TKE	Translational KE
Component.IFE	Inflator Energy
Component.DMP	Damping Energy
Component.PP	Ave Particle Pressure
Component.MAF	Mass flow rate via fabric
Component.MAV	Mass flow rate via vent
Component.MOF	Mass out via fabric
Component.MOV	Mass out via vent
Component.A_WO	Work Atmospheric
Component.U_WO	Work Useful
Component.NP	Num Particles
Component.AR	Total area
Component.PRP	+ve Pressure
Component.PRN	-ve Pressure
Component.HCE	Heat Convection Energy
Component.EV	Enhanced Vent Flag
Component.LE	Leakage Energy
Component.GAS	Gas Flow rate
Component.PVO	Por Volume
Component.PTE	Part Temperature
Component.UN	Unblocked Area
Component.BA	Blocked Area
Component.LK	Leakage
Component.FX	X Force

Component.FY	Y Force
Component.FZ	Z Force
Component.SPR	Static Pressure
Component.TPR	Total Pressure
Component.STEMP	Static Temperature
Component.TTEMP	Total Temperature
Component.VFR	Volume Flow Rate
Component.MFR	Mass Flow Rate
Component.HT	Heat Transfer
Component.HTR	Heat Transfer Rate
Component.HCR	Heat Convection Rate
Component.LER	Leakage Energy Rate
Component.TRE	Translational Energy
Component.X	X co-ordinate
Component.Y	Y co-ordinate
Component.Z	Z co-ordinate
Component.VX	X Velocity
Component.VY	Y Velocity
Component.VZ	Z Velocity
Component.VM	Velocity Magnitude

Constants for JOINT

Name	**Description**
Component.FX	X force
Component.FY	Y force
Component.FZ	Z force
Component.FM	Force Magnitude
Component.MX	Moment in X
Component.MY	Moment in Y
Component.MZ	Moment in Z
Component.MM	Moment Magnitude
Component.EN	Energy
Component.PHA	Phi angle
Component.PHDT	d(Phi)/dt
Component.PHS	Phi stiffness moment
Component.PHD	Phi damping moment
Component.PHT	Phi total moment
Component.THA	Theta angle
Component.THDT	d(Theta)/dt

Component.THS	Theta stiffness moment
Component.THD	Theta damping moment
Component.THT	Theta total moment
Component.PSA	Psi angle
Component.PSDT	d(Psi)/dt
Component.PSS	Psi stiffness moment
Component.PSD	Psi damping moment
Component.PST	Psi total moment
Component.AA	Alpha angle
Component.ADT	d(Alpha)/dt
Component.ALS	Alpha stiffness moment
Component.ALD	Alpha damping moment
Component.ALT	Alpha total moment
Component.BA	Beta angle
Component.BDT	d(Beta)/dt
Component.BES	Beta stiffness moment
Component.BED	Beta damping moment
Component.BET	Beta total moment
Component.GA	Gamma angle
Component.GDT	d(Gamma)/dt
Component.GSF	Gamma scale factor
Component.DX	X displacement
Component.DXDT	d(X)/dt
Component.DY	Y displacement
Component.DYDT	d(Y)/dt
Component.DZ	Z displacement
Component.DZDT	d(Z)/dt
Component.SFX	X stiffness force
Component.SFY	Y stiffness force
Component.SFZ	Z stiffness force
Component.DFX	X damping force
Component.DFY	Y damping force
Component.DFZ	Z damping force
Component.TFX	X total force
Component.TFY	Y total force
Component.TFZ	Z total force
Component.DP	P displacement
Component.DPDT	d(P)/dt
Component.DR	R displacement

Component.DRDT	d(R)/dt
Component.SFP	P stiffness force
Component.SFR	R stiffness force
Component.DFP	P damping force
Component.DFR	R damping force
Component.TFP	P total force
Component.TFR	R total force

Constants for X_SECTION

Name	**Description**
Component.XSEC_FX	X force
Component.XSEC_FY	Y force
Component.XSEC_FZ	Z force
Component.XSEC_FM	Force Magnitude
Component.XSEC_MX	Moment in X
Component.XSEC_MY	Moment in Y
Component.XSEC_MZ	Moment in Z
Component.XSEC_MM	Moment Magnitude
Component.XSEC_CX	X centroid coord
Component.XSEC_CY	Y centroid coord
Component.XSEC_CZ	Z centroid coord
Component.XSEC_A	Area of section

Constants for SUBSYSTEM

Name	**Description**
Component.GKE	Kinetic energy
Component.GIE	Internal energy
Component.GHG	Hourglass energy
Component.GKR	Kinetic Energy Ratio
Component.GIR	Internal Energy Ratio
Component.GMX	X momentum
Component.GMY	Y momentum
Component.GMZ	Z momentum
Component.MASS	Total mass
Component.GCM	Center of mass
Component.GXCM	X Center of mass
Component.GYCM	Y Center of mass

Component.GZCM	Z Center of mass
Component.GI11	Inertia Tensor Row11
Component.GI12	Inertia Tensor Row12
Component.GI13	Inertia Tensor Row13
Component.GI21	Inertia Tensor Row21
Component.GI22	Inertia Tensor Row22
Component.GI23	Inertia Tensor Row23
Component.GI31	Inertia Tensor Row31
Component.GI32	Inertia Tensor Row32
Component.GI33	Inertia Tensor Row33
Component.GI1	Principal inertia i11
Component.GI2	Principal inertia i22
Component.GI3	Principal inertia i33
Component.GP11	Principal Directions Row11
Component.GP12	Principal Directions Row12
Component.GP13	Principal Directions Row13
Component.GP21	Principal Directions Row21
Component.GP22	Principal Directions Row22
Component.GP23	Principal Directions Row23
Component.GP31	Principal Directions Row31
Component.GP32	Principal Directions Row32
Component.GP33	Principal Directions Row33

Constants for PART_GROUP

Name	**Description**
Component.GKE	Kinetic energy
Component.GIE	Internal energy
Component.GHG	Hourglass energy
Component.GTE	Total energy
Component.GMADD	Added mass

Constants for GEOMETRIC_CONTACT

Name	**Description**
Component.FX	X force
Component.FY	Y force
Component.FZ	Z force
Component.FM	Force Magnitude
Component.MX	Moment in X

Component.MY	Moment in Y
Component.MZ	Moment in Z
Component.MM	Moment Magnitude

Constants for NODAL_RB

Name	Description
Component.DX	X Displacement
Component.DY	Y Displacement
Component.DZ	Z Displacement
Component.DM	Displacement Magnitude
Component.VX	X Velocity
Component.VY	Y Velocity
Component.VZ	Z Velocity
Component.VM	Velocity Magnitude
Component.AX	X Acceleration
Component.AY	Y Acceleration
Component.AZ	Z Acceleration
Component.AM	Acceleration Magnitude
Component.CX	X co-ordinate
Component.CY	Y co-ordinate
Component.CZ	Z co-ordinate
Component.RDX	X rotation
Component.RDY	Y rotation
Component.RDZ	Z rotation
Component.RDM	Rotation Magnitude
Component.RVX	X rotational velocity
Component.RVY	Y rotational velocity
Component.RVZ	Z rotational velocity
Component.RVM	Rotation Vel Magnitude
Component.RAX	X rotational acceleration
Component.RAY	Y rotational acceleration
Component.RAZ	Z rotational acceleration
Component.RAM	Rotation Accel Magnitude
Component.D11	Direction Cosine 11
Component.D12	Direction Cosine 12
Component.D13	Direction Cosine 13
Component.D21	Direction Cosine 21
Component.D22	Direction Cosine 22
Component.D23	Direction Cosine 23

Component.D31	Direction Cosine 31
Component.D32	Direction Cosine 32
Component.D33	Direction Cosine 33
Component.LDX	Local X Displacement
Component.LDY	Local Y Displacement
Component.LDZ	Local Z Displacement
Component.LVX	Local X Velocity
Component.LVY	Local Y Velocity
Component.LVZ	Local Z Velocity
Component.LAX	Local X Acceleration
Component.LAY	Local Y Acceleration
Component.LAZ	Local Z Acceleration
Component.LRDX	Local X rotation
Component.LRDY	Local Y rotation
Component.LRDZ	Local Z rotation
Component.LRVX	Local X rotational vel
Component.LRVY	Local Y rotational vel
Component.LRVZ	Local Z rotational vel
Component.LRAX	Local X rotational accel
Component.LRAY	Local Y rotational accel
Component.LRAZ	Local Z rotational accel

Constants for WELD

Name	**Description**
Component.SW_F	Axial force
Component.SW_S	Shear force
Component.SW_FAIL	Failure
Component.SW_MF	Maximum Failure
Component.SW_LE	Length
Component.SW_TIME	Failure Time
Component.SW_TO	Torsion
Component.SW_MM	Moment Magnitude
Component.SW_FF	DC Failure Function
Component.SW_NF	Normal Failure
Component.SW_SF	Shear Failure
Component.SW_BF	Bending Failure
Component.SW_AREA	Spotweld Area

Constants for SPC

Name	Description
Component.SPC_FX	X Force
Component.SPC_FY	Y Force
Component.SPC_FZ	Z Force
Component.SPC_FM	Force Magnitude
Component.SPC_MX	Moment in X
Component.SPC_MY	Moment in Y
Component.SPC_MZ	Moment in Z
Component.SPC_MM	Moment Magnitude
Component.SPC_XTF	X Total Set Force
Component.SPC_YTF	Y Total Set Force
Component.SPC_ZTF	Z Total Set Force
Component.SPC_RF	Resultant Set Force
Component.SPC_XMF	X Total Model Force
Component.SPC_YMF	Y Total Model Force
Component.SPC_ZMF	Z Total Model Force
Component.SPC_RMF	Resultant Model Force

Constants for BOUNDARY

Name	Description
Component.FX	Applied X Force
Component.FY	Applied Y Force
Component.FZ	Applied Z Force
Component.FR	Applied Resultant force
Component.MX	Applied X Moment
Component.MY	Applied Y Moment
Component.MZ	Applied Z Moment
Component.MM	Applied Moment Magnitude
Component.EN	Energy from applied force

Constants for FSI

Name	Description
Component.PR	Pressure
Component.FX	X force
Component.FY	Y force

Component.FZ	Z force
Component.FM	Force Magnitude
Component.MP	Mass (Porous+Vent)
Component.ML	Mass (Leakage)
Component.LFX	Leakage X Force
Component.LFY	Leakage Y Force
Component.LFZ	Leakage Z Force
Component.LFM	Leakage Force Magnitude
Component.TEMP	Temperature
Component.TC	Temperature Change
Component.X	X co-ordinate
Component.Y	Y co-ordinate
Component.Z	Z co-ordinate
Component.SO	Cpld Solid ID

Constants for SPH

Name	**Description**
Component.DE	Density
Component.EXX	Strain in XX
Component.EYY	Strain in YY
Component.EZZ	Strain in ZZ
Component.EXY	Strain in XY
Component.EYZ	Strain in YZ
Component.EZX	Strain in ZX
Component.EFS	Effective Stress
Component.SXX	Stress in XX
Component.SYY	Stress in YY
Component.SZZ	Stress in ZZ
Component.SXY	Stress in XY
Component.SYZ	Stress in YZ
Component.SZX	Stress in ZX
Component.SM	Smoothing Length
Component.TEMP	Temperature
Component.ERXX	Strain in XX
Component.ERYY	Strain in YY
Component.ERZZ	Strain in ZZ
Component.ERXY	Strain in XY
Component.ERYZ	Strain in YZ
Component.ERZX	Strain in ZX

Constants for TRACER

Name	Description
Component.CX	X co-ordinate
Component.CY	Y co-ordinate
Component.CZ	Z co-ordinate
Component.CV	Current Vector
Component.VX	X Velocity
Component.VY	Y Velocity
Component.VZ	Z Velocity
Component.VM	Velocity Magnitude
Component.SXX	Stress in XX
Component.SYY	Stress in YY
Component.SZZ	Stress in ZZ
Component.SXY	Stress in XY
Component.SYZ	Stress in YZ
Component.SZX	Stress in ZX
Component.EPL	Effective Plastic Strain
Component.DE	Density
Component.RV	Relative Volume
Component.AC	Active

Constants for PULLEY

Name	Description
Component.PL_FT	Force
Component.PL_SL	Slip
Component.PL_SR	Slip Rate
Component.PL_AN	Wrap Angle

Constants for ICFD

Name	Description
Component.FPX	X Pressure Drag
Component.FPY	Y Pressure Drag
Component.FPZ	Z Pressure Drag
Component.FPM	Pressure Drag Magnitude
Component.FVX	X Viscous Drag
Component.FVY	Y Viscous Drag
Component.FVZ	Z Viscous Drag

Component.FVM	Viscous Drag Magnitude
Component.MPX	MX Pressure Drag
Component.MPY	MY Pressure Drag
Component.MPZ	MZ Pressure Drag
Component.MPM	Pressure Drag Magnitude
Component.MVX	MX Viscous Drag
Component.MVY	MY Viscous Drag
Component.MVZ	MZ Viscous Drag
Component.MVM	Viscous Drag Magnitude
Component.CX	X co-ordinate
Component.CY	Y co-ordinate
Component.CZ	Z co-ordinate
Component.CV	Current Vector
Component.VX	X Velocity
Component.VY	Y Velocity
Component.VZ	Z Velocity
Component.VM	Velocity Magnitude
Component.AVX	X AVelocity
Component.AVY	Y AVelocity
Component.AVZ	Z AVelocity
Component.AVM	AVelocity Magnitude
Component.PR	Pressure
Component.PA	Average Pressure
Component.DE	Density
Component.VTX	X Vorticity
Component.VTY	Y Vorticity
Component.VTZ	Z Vorticity
Component.VTM	Vorticity Magnitude
Component.QC	Q Criterion
Component.VC	Viscosity
Component.VT	Viscous Turbulence
Component.LS	Level Set Function
Component.A	Alpha
Component.TEMP	Temperature
Component.TAA	Temp Area Average
Component.TSA	Temp Sum Average
Component.TEH	Average Heat Flux
Component.AR	Total Area
Component.HTC	Heat Transfer Coeff

Component.VO	Volume
Component.FR	Flow Rate

Constants for CESE

Name	Description
Component.CX	X co-ordinate
Component.CY	Y co-ordinate
Component.CZ	Z co-ordinate
Component.CV	Current Vector
Component.VX	X Velocity
Component.VY	Y Velocity
Component.VZ	Z Velocity
Component.VM	Velocity Magnitude
Component.VTX	X Vorticity
Component.VTY	Y Vorticity
Component.VTZ	Z Vorticity
Component.VTM	Vorticity Magnitude
Component.DE	Density
Component.PR	Pressure
Component.TEMP	Temperature
Component.FPX	X Pressure Force
Component.FPY	Y Pressure Force
Component.FPZ	Z Pressure Force
Component.FPM	Pressure Force Magnitdue
Component.FVX	X Viscous Force
Component.FVY	Y Viscous Force
Component.FVZ	Z Viscous Force
Component.FVM	Viscous Force Magnitude
Component.AR	Total Area

Constants for EM

Name	Description
Component.CX	X co-ordinate
Component.CY	Y co-ordinate
Component.CZ	Z co-ordinate
Component.CV	Current Vector
Component.ECX	X Current
Component.ECY	Y Current

Component.ECZ	Z Current
Component.ECM	Current Magnitude
Component.BFDX	X BField
Component.BFDY	Y BField
Component.BFDZ	Z BField
Component.BFDM	BField Magnitude
Component.AFX	X AField
Component.AFY	Y AField
Component.AFZ	Z AField
Component.AFM	AField Magnitude
Component.S	Sigma
Component.MUR	Relative Permeability
Component.JHR	Joule Heating Rate
Component.LOFX	X Lorentz Force
Component.LOFY	Y Lorentz Force
Component.LOFZ	Z Lorentz Force
Component.LOFM	Lorentz Force Magnitude
Component.EFX	X EField
Component.EFY	Y EField
Component.EFZ	Z EField
Component.EFM	EField Magnitude
Component.SP	Scalar Potential
Component.ECV	Voltage
Component.ECC	Charge
Component.ECCT	Current
Component.ECRD	Circuit Resistance
Component.ECRJ	Equivalent Resistance
Component.ECI	Inductance
Component.ECM1	Mutual Inductance 1
Component.ECM2	Mutual Inductance 2
Component.ECM3	Mutual Inductance 3
Component.ECDV	Voltage
Component.ECDC	Charge
Component.ECDT	Current
Component.ECDE	Total Energy
Component.PLFX	X Lorentz Force
Component.PLFY	Y Lorentz Force
Component.PLFZ	Z Lorentz Force
Component.PLFM	M Lorentz Force

Component.PJHE	Joule Heating Energy
Component.PMAG	Magnetic Energy
Component.PKIN	Kinetic Energy
Component.PPLA	Plastic Energy
Component.EIV	Voltage
Component.EICT	Current
Component.ECRC	Contact Current
Component.ECRR	Contact Resistance
Component.ECRA	Contact Area
Component.EBV	Voltage
Component.EBC	Current
Component.EBA	Area
Component.ECT	Current
Component.ERD	Contact Resistance
Component.POW	Power
Component.ENE	Energy
Component.TVO	TotVoltage
Component.OCV	OCV
Component.DVO	DampVoltage
Component.RCT	Current
Component.SOC	SOC
Component.SOF	SOCFunc
Component.SOS	SOCShift
Component.SOM	SOCSum
Component.RR0	R0
Component.R10	R10
Component.C10	C10
Component.TEM	Temp
Component.CNM	Ckt Number
Component.ERVC	Volume Current
Component.ERSC	Surface Current
Component.ERVM	Magnetic Field
Component.RUN	Run timestep
Component.CFL	Condition timestep
Component.RBC	Ratio
Component.VC2	VC2
Component.VC3	VC3
Component.R20	R20
Component.R30	R30

Component.C20	C20
Component.C30	C30
Component.OHP	Ohm Heat Power
Component.RHP	Reversible Heat Power
Component.ECP	Equivalent Capacity Power
Component.OHE	Ohm heat energy
Component.RHE	Reversible heat energy
Component.ECE	Equivalent Capacity energy
Component.ESE	Equivalent storage energy
Component.ECJH	Ext ckt Joule Heating
Component.ECME	Ext ckt Magnetic Energy
Component.ECCE	Ext ckt Capacitor Energy
Component.MJH	Mesh conductor Joule Heating
Component.MME	Mesh conductor Mag Energy
Component.AME	Air Magnetic Energy
Component.TEE	Total EM Energy
Component.TPE	Total Plastic Energy
Component.TKE	Total kinetic Energy
Component.MSR	Maximum short resistance
Component.NSC	Number of short circuits
Component.TNC	Total number of circuits
Component.TSR	Total short resistance
Component.MXR	Maximum resistance
Component.SHC	Short circuits
Component.TOC	Total circuits
Component.TOR	Total resistance
Component.ARS	Area short

Constants for PBLAST

Name	**Description**
Component.AIE	Air Internal Energy
Component.DPIE	Detn Product IE
Component.OIE	Outside Domain IE
Component.ATE	Air Translational E
Component.DPTE	Detn Product Trans E
Component.OTE	Outside Domain Trans E
Component.APR	Air Pressure
Component.DPPR	Detn Product Pressure
Component.RPR	Resultant Pressure

Component.AR	Surface Area
Component.AFX	Air X Force
Component.AFY	Air Y Force
Component.AFZ	Air Z Force
Component.DPFX	Detn Product X Force
Component.DPFY	Detn Product Y Force
Component.DPFZ	Detn Product Z Force
Component.RFX	Resultant X Force
Component.RFY	Resultant Y Force
Component.RFZ	Resultant Z Force

Constants for PRTUBE

Name	Description
Component.AR	Cross section area
Component.DE	Density
Component.PR	Pressure
Component.VEL	Velocity

Constants for BEARING

Name	Description
Component.FX	X Force
Component.FY	Y Force
Component.FZ	Z Force
Component.MX	X Moment
Component.MY	Y Moment
Component.MZ	Z Moment
Component.DX	X Displacement
Component.DY	Y Displacement
Component.DZ	Z Displacement
Component.AX	X Angle
Component.AY	Y Angle
Component.AZ	Z Angle
Component.LFX	Local X Force
Component.LFY	Local Y Force
Component.LFZ	Local Z Force
Component.LMX	Local X Moment
Component.LMY	Local Y Moment
Component.LMZ	Local Z Moment

Component.LDX	Local X Displacement
Component.LDY	Local Y Displacement
Component.LDZ	Local Z Displacement
Component.LAX	Local X Angle
Component.LAY	Local Y Angle
Component.LAZ	Local Z Angle

Constants for CURVOUT

Name	Description
Component.COUT	CURVOUT

Constants for DEMRCF

Name	Description
Component.DCFX	X force
Component.DCFY	Y force
Component.DCFZ	Z force
Component.DCFM	Force Mag
Component.DCMX	X moment
Component.DCMY	Y moment
Component.DCMZ	Z moment
Component.DCMM	Moment Mag
Component.DCM	Mass

Constants for index

Name	Description
Component. js_label	button_label

Constant class

The Constant class defines various constants. More...

The T/HIS JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Constant constants

Constants for Surface

Name	Description
Constant.BOTTOM	Bottom shell surface
Constant.MIDDLE	Middle shell surface
Constant.TOP	Top shell surface

Constants for THISLinkStatus

Name	Description
Constant.ALL_UNDOCKED	T/HIS linked to D3PLOT and all undocked
Constant.DOCKED	T/HIS linked to D3PLOT and docked
Constant.MENUS_UNDOCKED	T/HIS linked to D3PLOT and menus undocked
Constant.STANDALONE	T/HIS in standalone mode

Detailed Description

The Constant class gives you access to various constants that are used in the T/HIS API. They are defined in the Constant class so the global namespace is not polluted. See the documentation below for more details.

Curve class

The Curve class gives you access to curves in T/HIS. More...

The T/HIS JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- AddFlaggedToGraph(flag*[Flag]*, graph*[int]*)
- Copy(source*[integer]*, target*[integer]*)
- Delete(curve*[integer]*)
- DeleteFlagged(flag*[Flag]*)
- Exists(curve*[integer]*)
- First()
- FirstFreeID()
- FirstID()
- FlagAll(flag*[integer]*)
- GetFlagged(flag*[Flag]*)
- GetFromID(ID*[integer]*)
- GetFromTag(TAG*[string]*)
- HighestID()
- Pick(prompt*[string]*, modal (optional)*[boolean]*)
- RemoveFlaggedFromGraph(flag*[Flag]*, graph*[int]*)
- Select(flag*[integer]*, prompt*[string]*, modal (optional)*[boolean]*)
- UnflagAll(flag*[integer]*)

Member functions

- AddPoint(xvalue*[real]*, yvalue*[real]*)
- AddToGraph(graph*[int]*)
- ClearFlag(flag*[integer]*)
- DeletePoint(ipt*[integer]*)
- Flagged(flag*[integer]*)
- Freeze(graph*[integer]*, Freeze option*[integer]*)
- GetPoint(row*[integer]*)
- InsertPoint(ipt*[integer]*, xvalue*[real]*, yvalue*[real]*, position*[integer]*)
- Next()
- Previous()
- RemoveFromGraph(graph*[int]*)
- SetFlag(flag*[integer]*)
- SetPoint(ipt*[integer]*, xvalue*[real]*, yvalue*[real]*)
- Update()
- YatX(xvalue*[real]*)

Curve constants

Name	Description
Curve.AFTER	Insertion of curve data option.
Curve.BEFORE	Insertion of curve data option.
Curve.Y1_AXIS	Y1 axis option.
Curve.Y2_AXIS	Y2 axis option.

Curve properties

Name	Type	Description
average (read only)	real	Curve average value
colour	integer	The Colour of the curve
directory	string	Directory the curve came from
entity_id	integer	The ID of the entity that the curve was generated from.
entity_type	integer	The Entity type that the curve was generated from
file	string	Filename the curve came from
hic (read only)	real	Curve HIC value - returns 0.0 if the HIC hasn't been calculated
hic_tmax (read only)	real	End of HIC time windows - returns 0.0 if the HIC hasn't been calculated
hic_tmin (read only)	real	Start of HIC time windows - returns 0.0 if the HIC hasn't been calculated
hicd (read only)	real	Curve HIC(d) value - returns 0.0 if the HIC(d) hasn't been calculated
hicd_tmax (read only)	real	End of HIC(d) time windows - returns 0.0 if the HIC(d) hasn't been calculated
hicd_tmin (read only)	real	Start of HIC(d) time windows - returns 0.0 if the HIC(d) hasn't been calculated
id (read only)	integer	Curve ID
is_null (read only)	integer	Returns 1 if the curve is NULL
label	string	Curve label
model	integer	The ID of the model that a curve was read from.
npoints (read only)	integer	Number of curve points
regr_rsq (read only)	real	Pearson's R^2 value for regression curve, returns 0.0 if the curve has not come from the regression operation.
regr_sdgrad (read only)	real	Standard deviation of the linear regression gradient value, returns 0.0 if the curve has not come from linear regression.
regr_sdicpt (read only)	real	Standard deviation of the linear regression intercept value, returns 0.0 if the curve has not come from linear regression.
regr_sdyx (read only)	real	Standard deviation of the linear regression values 'y = bx + c', returns 0.0 if the curve has not come from linear regression.
rms (read only)	real	Curve RMS value
style	integer	The LineStyle used to draw the curve
symbol	integer	The Symbol style for a curve
tag	string	Curve tag. If a FAST-TCF script is running then this is the FAST-TCF tag
title	string	Curve title
tms (read only)	real	3ms Clip value - returns 0.0 if the 3ms Clip value hasn't been calculated
tms_tmax (read only)	real	End of 3ms clip time windows - returns 0.0 if the 3ms Clip hasn't been calculated
tms_tmin (read only)	real	Start of 3ms clip time windows - returns 0.0 if the 3ms Clip hasn't been calculated
unit_system	integer	The Curve UnitSystem

width	integer	The LineWidth used to draw the curve
x_at_ymax (read only)	real	X axis value at the Y axis maximum
x_at_ymin (read only)	real	X axis value at the Y axis minimum
x_axis_label	string	Curve X axis label
x_axis_unit	integer	The X axis Units
xmax (read only)	real	X axis maximum value
xmin (read only)	real	X axis minimum value
y_axis	integer	The Y axis used by the curve (Curve.Y1_AXIS or Curve.Y2_AXIS). (Note to turn on the display of the Y2 axis you need to set the show_y2axis on the graph to Graph.ON)
y_axis_label	string	Curve Y axis label
y_axis_unit	integer	The Y axis Units
ymax (read only)	real	Y axis maximum value
ymin (read only)	real	Y axis minimum value

Detailed Description

The Curve class allows you to create, modify, edit and manipulate curves. See the documentation below for more details.

Constructor

new Curve(lcid*[integer]*, tag (optional)*[string]*, Line label (optional)*[string]*, X-axis label (optional)*[string]*, Y-axis label (optional)*[string]*)

Description

Create a new Curve object. The curve will be added to all the currently active graphs.

Arguments

- **lcid** (integer)

Curve number

- **tag (optional)** (string)

Tag used to reference the curve in FAST-TCF scripts

- **Line label (optional)** (string)

Line label for the curve

- **X-axis label (optional)** (string)

X-axis label for the curve

- **Y-axis label (optional)** (string)

Y-axis label for the curve

Returns

Curve object

Return type

Curve

Example

To create a new curve with label 200

```
var l = new Curve(200);
```

Details of functions

AddFlaggedToGraph(flag*[Flag]*, graph*[int]*) [static]

Description

Adds flagged curves to a graph.

Arguments

- **flag** (Flag)

Flag to check on the curve

- **graph** (int)

Graph to add the curve to. If undefined then the curve is added to all graphs.

This argument can be repeated if required
Alternatively a single array argument containing the multiple values can be given

Returns

No return value.

Example

To add curves flagged with flag f to graphs 1 and 3:

```
Curve.AddFlaggedToGraph(f,1,3);
```

To add curves flagged with flag to all graphs:

```
Curve.AddFlaggedToGraph(f);
```

AddPoint(xvalue*[real]*, yvalue*[real]*)

Description

Adds a point at the end of the curve.

Arguments

- **xvalue** (real)

The x value of the point.

- **yvalue** (real)

The y value of the point.

Returns

No return value.

Example

To add the point x=3.5, y=5.5 to curve l:

```
l.AddPoint(3.5,5.5);
```

AddToGraph(graph*[int]*)

Description

Adds a curve to a graph.

Arguments

- **graph** (int)

Graph to add the curve to. If undefined then the curve is added to all graphs.

This argument can be repeated if required
Alternatively a single array argument containing the multiple values can be given

Returns

No return value.

Example

To add a curve (c) to graphs 1 and 3:

```
c.AddToGraph(1,3);
```

To add a curve (c) to all graphs:

```
c.AddToGraph();
```

ClearFlag(flag*[integer]*)

Description

Clears a flag on the curve.

Arguments

- **flag** (integer)

Flag to clear on the curve

Returns

No return value

Example

To clear flag f for curve l:

```
l.ClearFlag(f);
```

Copy(source*[integer]*, target*[integer]*) [static]

Description

Copies a curve.

Arguments

- **source** (integer)

ID of curve to copy from

- **target** (integer)

ID of curve to copy to

Returns

No return value

Example

To copy curve 1 to curve 4:

```
Curve.Copy(1,4);
```

To copy curve a to curve b,

```
Curve.Copy(a.id,b.id);
```

Delete(curve*[integer]*) [static]

Description

Deletes a curve

Arguments

- **curve** (integer)

ID of curve to delete

Returns

No return value

Example

To delete curve n

```
Curve.Delete(n);
```

DeleteFlagged(flag*[Flag]*) [static]

Description

Deletes flagged curves

Arguments

- **flag** (Flag)

Flag to check on the curve

Returns

No return value

Example

To delete curves flagged with flag f

```
Curve.DeleteFlagged(f);
```

DeletePoint(ipt*[integer]*)

Description

Deletes a point in a curve. The input for the point number should start at 1 for the 1st point not zero.

Arguments

- **ipt** (integer)

The point you want to insert the data before or after.

Returns

No return value.

Example

To delete the 3rd point in curve l:

```
l.DeletePoint(3);
```

Exists(curve*[integer]*) [static]

Description

Checks if a curve exists

Arguments

- **curve** (integer)

ID of curve to check

Returns

TRUE if the curve exists, otherwise FALSE

Return type

Boolean

Example

To check if a curve n exists

```
var exists = Curve.Exists(n);
```

First() [static]

Description

Returns the first curve.

Arguments

No arguments

Returns

Curve object (or null if there are no more curves in the model).

Return type

Curve

Example

To get the 1st curve

```
var curve = Curve.First();
```

FirstFreeID() [static]

Description

Returns the ID of the first free curve.

Arguments

No arguments

Returns

ID of first unsued curve.

Return type

Number

Example

To get the ID of the first free curve:

```
var curve = Curve.FirstFreeID();
```

FirstID() [static]

Description

Returns the ID of the first curve.

Arguments

No arguments

Returns

ID of the first curve defined.

Return type

Number

Example

To get the 1st curve

```
var curve = Curve.FirstID();
```

FlagAll(flag*[integer]*) [static]

Description

Flags all of the curves with a defined flag

Arguments

- **flag** (integer)

Flag to set on the curves

Returns

No return value

Example

To flag all of the curves with flag f:

```
Curve.FlagAll(f);
```

Flagged(flag*[integer]*)

Description

Checks if the curve is flagged or not.

Arguments

- **flag** (integer)

Flag to check on the curve

Returns

true if flagged, false if not.

Return type

Boolean

Example

To check if curve d has flag f set on it:

```
if (d.Flagged(f) ) do_something...
```

Freeze(graph*[integer]*, Freeze option*[integer]*)

Description

Freezes an unblanked curve on one or all graphs.

Arguments

- **graph** (integer)

Graph number to freeze curve on or 0 for all graphs.

- **Freeze option** (integer)

No argument or 1 to freeze the curve, 0 to unfreeze.

Returns

No return value

Example

To freeze a curve c on graph 3:

```
c.Freeze(3,1)
```

GetFlagged(flag*[Flag]*) [static]

Description

Returns an array of all curves flagged with a given flag.

Arguments

- **flag** (Flag)

Flag for which to return flagged objects.

Returns

Array of Curve objects (or null if no curves are flagged)

Return type

Array

Example

To get the curves flagged with flag f:

```
var curve_array = Curve.GetFlagged(f);
```

GetFromID(ID*[integer]*) [static]

Description

Returns the curve object for a curve ID.

Arguments

- **ID** (integer)

ID of curve to return object for

Returns

Curve object (or null if the curve does not exist.

Return type

Curve

Example

To get the curve n

```
var curve = Curve.GetFromID(n);
```

GetFromTag(TAG*[string]*) [static]

Description

Finds a curve from it’s Tag. This function is only available when running a Javascript from within a FAST-TCF script

Arguments

- **TAG** (string)

TAG of curve to return object for

Returns

Curve object (or null if there are no free curves).

Return type

Curve

Example

To get the curve with a tag "tag"

```
var curve = Curve.GetFromTag(tag);
```

GetPoint(row*[integer]*)

Description

Returns x and y data for a point in a curve. The input for the point number should start at 1 for the 1st point not zero. In the array returned array[0] contains the x axis value and array[1] contains the y-axis value.

Arguments

- **row** (integer)

The point you want the data for.

Returns

An array of numbers containing the x and y point values

Return type

Number

Example

To get the curve data for the 3rd point for curve l:

```
if (l.npoints >= 3)
{
    var point_data = l.GetPoint(3);
}
```

HighestID() [static]

Description

Returns the ID of the highest curve currently being used

Arguments

No arguments

Returns

ID of highest curve currently being used.

Return type

Number

Example

To get the highest curve ID

```
var id= Curve.HighestID();
```

InsertPoint(ipt*[integer]*, xvalue*[real]*, yvalue*[real]*, position*[integer]*)

Description

Inserts a new point before or after the specified point.

Arguments

- **ipt** (integer)

The point you want to insert the data before or after.

- **xvalue** (real)

The x value of the point.

- **yvalue** (real)

The y value of the point.

- **position** (integer)

Specify either before or after the selected pioint. Use ’Curve.BEFORE’ for before, and ’Curve.AFTER’ for after.

Returns

No return value.

Example

To insert the values after the 3rd row to x=3, y=5 for curve l:

```
l.InsertPoint(3, 3, 5, Curve.AFTER);
```

Next()

Description

Returns the next curve in the model.

Arguments

No arguments

Returns

Curve object (or null if there are no more curves in the model).

Return type

Curve

Example

To get the curve in model m after curve l:

```
var curve = l.Next();
```

Pick(prompt*[string]*, modal (optional)*[boolean]*) [static]

Description

Picks a single curve.

Arguments

- **prompt** (string)

Text to display as a prompt to the user

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in T/HIS until this window is dismissed). If omitted the selection will be modal.

Returns

Curve object (or null if the user cancels the pick operation).

Return type

Curve

Example

To pick a curve, giving the prompt 'Pick curve':

```
var curve = Curve.Pick('Pick curves');
```

Previous()

Description

Returns the previous curve in the model.

Arguments

No arguments

Returns

Curve object (or null if there are no more curves in the model).

Return type

Curve

Example

To get the curve in model m before this one:

```
var curve = curve.Previous();
```

RemoveFlaggedFromGraph(flag*[Flag]*, graph*[int]*) [static]

Description

Removes flagged curves from a graph.

Arguments

- **flag** (Flag)

Flag to check on the curve

- **graph** (int)

Graph to remove the curve from. If undefined then the curve is removed from all graphs.

This argument can be repeated if required
Alternatively a single array argument containing the multiple values can be given

Returns

No return value.

Example

To remove curves flagged with flag f from graphs 1 and 3:

```
Curve.RemoveFlaggedFromGraph(f,1,3);
```

To remove curves flagged with flag f from all graphs:

```
Curve.RemoveFlaggedFromGraph(f);
```

RemoveFromGraph(graph*[int]*)

Description

Removes a curve from a graph.

Arguments

- **graph** (int)

Graph to remove the curve from, If undefined then the curve is removed from all graphs.

This argument can be repeated if required
Alternatively a single array argument containing the multiple values can be given

Returns

No return value.

Example

To remove a curve (c) from graphs 1 and 3:

```
c.RemoveFromGraph(1,3);
```

To remove a curve (c) from all graphs:

```
c.RemoveFromGraph();
```

Select(flag*[integer]*, prompt*[string]*, modal (optional)*[boolean]*) [static]

Description

Allows the user to select curves.

Arguments

- **flag** (integer)

Flag to use when selecting curves

- **prompt** (string)

Text to display as a prompt to the user

- **modal (optional)** (boolean)

If selection is modal (blocks the user from doing anything else in T/HIS until this window is dismissed). If omitted the selection will be modal.

Returns

Number of items selected or null if menu cancelled

Return type

Number

Example

To select curves, flagging those selected which flag f, giving the prompt ’Select curves’:

```
var num = Curve.Select(f, ’Select curves’);
```

SetFlag(flag*[integer]*)

Description

Sets a flag on the curve.

Arguments

- **flag** (integer)

Flag to set on the curve

Returns

No return value

Example

To set flag f for curve l:

```
l.SetFlag(f);
```

SetPoint(ipt*[integer]*, xvalue*[real]*, yvalue*[real]*)

Description

Sets the x and y values for a specified point in a curve.

Arguments

- **ipt** (integer)

The point to set the data for.

- **xvalue** (real)

The x value of the point.

- **yvalue** (real)

The y value of the point.

Returns

No return value.

Example

To set the values for the 3rd point to x=3, y=5 for curve l:

```
l.SetPoint(3, 3, 5);
```

UnflagAll(flag*[integer]*) [static]

Description

Unsets a defined flag on all of the curves.

Arguments

- **flag** (integer)

Flag to unset on the curves

Returns

No return value

Example

To unset the flag f on all of the curves:

```
Curve.UnflagAll(f);
```

Update()

Description

Updates a curve properties (min,max, average values etc).

Arguments

No arguments

Returns

No return value.

Example

To update the properties of curve l:

```
l.Update();
```

YatX(xvalue*[real]*)

Description

Returns the y value of the curve at a given x value, interpolating if requested x value lies between data points.

Arguments

- **xvalue** (real)

The x value.

Returns

Y value

Return type

real

Example

To get the y value of curve c when x=1.4:

```
var y = c.YatX(1.4);
```

Datum class

The Datum class gives you access to datums in T/HIS. More...

The T/HIS JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Delete(datum*[string]*)
- Exists(datum*[string]*)
- First()
- GetFromAcronym(datum*[string]*)

Member functions

- AddToGraph(graph*[int]*)
- IsOnGraph(graph*[int]*)
- Next()
- RemoveFromGraph(graph*[int]*)

Datum constants

Name	Description
Datum.CONSTANT_X	Constant X type datum.
Datum.CONSTANT_Y	Constant Y type datum.
Datum.CONSTANT_Y2	Constant Y2 type datum.
Datum.FILL_ABOVE_BELOW	Fill datum above and below.
Datum.FILL_RIGHT_LEFT	Fill datum right and left.
Datum.LABEL_10_POINT	Label font size 10.
Datum.LABEL_12_POINT	Label font size 12.
Datum.LABEL_14_POINT	Label font size 14.
Datum.LABEL_18_POINT	Label font size 16.
Datum.LABEL_24_POINT	Label font size 24.
Datum.LABEL_8_POINT	Label font size 8.
Datum.LABEL_ABOVE_CENTRE	Label position above centre.
Datum.LABEL_ABOVE_LEFT	Label position above left.
Datum.LABEL_ABOVE_RIGHT	Label position above right.
Datum.LABEL_AUTOMATIC	Label automatic font size.
Datum.LABEL_BELOW_CENTRE	Label position below centre.
Datum.LABEL_BELOW_LEFT	Label position below left.
Datum.LABEL_BELOW_RIGHT	Label position below right.

Datum.LABEL_BOTTOM_LEFT	Label position bottom left.
Datum.LABEL_BOTTOM_RIGHT	Label position bottom right.
Datum.LABEL_COURIER_BOLD	Label Courier bold font.
Datum.LABEL_COURIER_MEDIUM	Label Courier medium font.
Datum.LABEL_DEFAULT	Label default font.
Datum.LABEL_HELVETICA_BOLD	Label Helvetica bold font.
Datum.LABEL_HELVETICA_MEDIUM	Label Helvetica medium font.
Datum.LABEL_HORIZONTAL	Label horizonal orientation.
Datum.LABEL_MIDDLE_LEFT	Label position middle left.
Datum.LABEL_MIDDLE_RIGHT	Label position middle right.
Datum.LABEL_NONE	No label.
Datum.LABEL_TIMES_BOLD	Label Times bold font.
Datum.LABEL_TIMES_MEDIUM	Label Times medium font.
Datum.LABEL_TOP_LEFT	Label position top left.
Datum.LABEL_TOP_RIGHT	Label position top right.
Datum.LABEL_VERTICAL	Label vertical orientation.
Datum.POINTS	Points type datum.

Datum properties

Name	Type	Description
acronym	string	Datum acronym
fill_colour_above	Colour	The colour above the datum line
fill_colour_below	Colour	The colour below the datum line
fill_colour_between	Colour	The colour in between the datum line and the optional second datum line
fill_colour_left	Colour	The colour left of the datum line
fill_colour_right	Colour	The colour right of the datum line
fill_type	integer	The fill type. Can be Datum.FILL_ABOVE_BELOW, Datum.FILL_RIGHT_LEFT. Note that this can only be changed if the datum is of the type Datum.POINTS.
label	string	Datum label
label2	string	Label for optional 2nd datum line
label_colour	Colour	The colour of the datum label
label_font	integer	The label font. Can be Datum.LABEL_DEFAULT, Datum.LABEL_HELVETICA_BOLD, Datum.LABEL_HELVETICA_MEDIUM, Datum.LABEL_TIMES_BOLD, Datum.LABEL_TIMES_MEDIUM, Datum.LABEL_COURIER_BOLD, Datum.LABEL_COURIER_MEDIUM
label_orientation	integer	The orientation of the label. Can be Datum.LABEL_HORIZONTAL, Datum.LABEL_VERTICAL

label_position	integer	The label position. Can be Datum.LABEL_NONE, Datum.LABEL_ABOVE_CENTRE, Datum.LABEL_ABOVE_LEFT, Datum.LABEL_ABOVE_RIGHT, Datum.LABEL_BELOW_CENTRE, Datum.LABEL_BELOW_LEFT, Datum.LABEL_BELOW_RIGHT, Datum.LABEL_MIDDLE_LEFT, Datum.LABEL_TOP_LEFT, Datum.LABEL_BOTTOM_LEFT, Datum.LABEL_MIDDLE_RIGHT, Datum.LABEL_TOP_RIGHT, Datum.LABEL_BOTTOM_RIGHT
label_size	integer	The label font size. Can be Datum.LABEL_AUTOMATIC, Datum.LABEL_8_POINT, Datum.LABEL_10_POINT, Datum.LABEL_12_POINT, Datum.LABEL_14_POINT, Datum.LABEL_18_POINT, Datum.LABEL_24_POINT,
line_colour	Colour	The colour of the datum line
line_style	LineStyle	The line style used to draw the datum line
line_width	LineWidth	The line width used to draw the datum line

Detailed Description

The Datum class allows you to create and manipulate datums. See the documentation below for more details.

Constructor

new Datum(acronym*[string]*, type*[integer]*, value*[real or array of reals]*, second value (optional)*[real]*)

Description

Create a new Datum object. The datum will be added to all the currently active graphs.

Arguments

- **acronym** (string)

Datum acronym

- **type** (integer)

Specify type of datum line. Can be Datum.CONSTANT_X, Datum.CONSTANT_Y, Datum.CONSTANT_Y2, Datum.POINTS

- **value** (real or array of reals)

Value for Datum.CONSTANT_X, Datum.CONSTANT_Y or Datum.CONSTANT_Y2 type Datum. If it is a Datum.POINTS type Datum then this should be an array of X, Y pairs or a curve ID to copy points from.

- **second value (optional)** (real)

Second constant value for use with constant X,Y or Y2 datums and can optionally be provided

Returns

Datum object

Return type

Datum

Example

To create a new datum with acronym my_datum and a constant Y value of 100

```
var d = new Datum("my_datum", Datum.CONSTANT_Y, 100);
```

To create a new datum with acronym my_datum and some X, Y points

```
var points = new Array(6);
points[0] = 0.0;
points[1] = 10.0;
points[2] = 1.0;
points[3] = 15.0;
points[4] = 2.0;
points[5] = 17.0;
var d = new Datum("my_datum", Datum.POINTS, points);
```

Details of functions

AddToGraph(graph*[int]*)

Description

Adds a datum to a graph.

Arguments

- **graph** (int)

Graph to add the datum to. If undefined then the datum is added to all graphs.

This argument can be repeated if required
Alternatively a single array argument containing the multiple values can be given

Returns

No return value.

Example

To add a datum (d) to graphs 1 and 3:

```
d.AddToGraph(1, 3);
```

To add a datum (d) to all graphs:

```
d.AddToGraph();
```

Delete(datum*[string]*) [static]

Description

Deletes a datum

Arguments

- **datum** (string)

Acronym of datum to delete

Returns

No return value

Example

To delete datum "my_datum"

```
Datum.Delete("my_datum");
```

Exists(datum*[string]*) [static]

Description

Checks if a datum exists

Arguments

- **datum** (string)

Acronym of datum to check

Returns

TRUE if the datum exists, otherwise FALSE

Return type

Boolean

Example

To check if a datum "my_datum" exists

```
var exists = Datum.Exists("my_datum");
```

First() [static]

Description

Returns the first datum.

Arguments

No arguments

Returns

Datum object (or null if there are no datum in the model).

Return type

Datum

Example

To get the 1st datum

```
var d = Datum.First();
```

GetFromAcronym(datum*[string]*) [static]

Description

Returns the datum object for a datum acronym.

Arguments

- **datum** (string)

Acronym of datum to return object for

Returns

Datum object (or null if the datum does not exist).

Return type

Datum

Example

To get the datum "my_datum"

```
var d = Datum.GetFromAcronym("my_datum");
```

IsOnGraph(graph*[int]*)

Description

Returns whether a datum is on a graph.

Arguments

- **graph** (int)

Graph id

Returns

true if it is on the graph, false otherwise

Return type

Boolean

Example

To check if datum (d) is on graph 3:

```
d.IsOnGraph(3);
```

Next()

Description

Returns the next datum in the model.

Arguments

No arguments

Returns

Datum object (or null if there are no more datums in the model).

Return type

Datum

Example

To get the next datum after datum d:

```
var datum = d.Next();
```

RemoveFromGraph(graph*[int]*)

Description

Removes a datum from a graph.

Arguments

- **graph** (int)

Graph to remove the datum from. If undefined then the datum is removed from all graphs.

This argument can be repeated if required
Alternatively a single array argument containing the multiple values can be given

Returns

No return value.

Example

To remove a datum (d) from graphs 1 and 3:

```
d.RemoveFromGraph(1,3);
```

To remove a datum (d) from all graphs:

```
d.RemoveFromGraph();
```

Entity class

The Entity class contains constants relating to Entity types. More...

The T/HIS JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Entity constants

Name	Description
Entity.AIRBAG	AIRBAG entity code (for all airbag related entities)
Entity.AIRBAG_CHAMBER_DATA	AIRBAG CHAMBER DATA entity code
Entity.AIRBAG_CPM_PART_DATA	AIRBAG CPM PART DATA entity code
Entity.AIRBAG_CPM_SENSORS	AIRBAG CPM SENSORS entity code
Entity.AIRBAG_CV_PART_DATA	AIRBAG CV PART DATA entity code
Entity.AIRBAG_DATA	AIRBAG DATA entity code
Entity.AIRBAG_PART_DATA	AIRBAG PART DATA entity code
Entity.BEAM	BEAM entity code
Entity.BEAM_DISCRETE	DISCRETE BEAM entity code
Entity.BEAM_NORMAL	NORMAL BEAM entity code
Entity.BEARING	BEARING entity code
Entity.BOUNDARY	BOUNDARY entity code
Entity.BOUNDARY_DIS_NODAL_LOAD	DISCRETE NODAL LOAD entity code
Entity.BOUNDARY_DIS_RBODY_LOAD	DISCRETE RIGID BODY LOAD entity code
Entity.BOUNDARY_PRES_NODAL_LOAD	PRESSURE NODAL LOAD entity code
Entity.BOUNDARY_VEL_NODAL_LOAD	VELOCITY NODAL LOAD entity code
Entity.BOUNDARY_VEL_RBODY_LOAD	VELOCITY RIGID BODY LOAD entity code
Entity.CESE	CESE entity code
Entity.CESE_DRAG_DATA	CESE FSI DRAG DATA entity code
Entity.CESE_NODE_DATA	CESE NODE DATA entity code
Entity.CESE_POINT_DATA	CESE POINT DATA entity code
Entity.CESE_SEGMENT_DATA	CESE SEGMENT SET DATA entity code
Entity.CONTACT	CONTACT entity code
Entity.CONTACT_ENERGIES	CONTACT ENERGIES entity code

Entity.CONTACT_FORCES	CONTACT FORCES entity code
Entity.CURVOUT	CURVOUT entity code
Entity.DEMRCF	DEMRCF entity code
Entity.EM	EM entity code
Entity.EM_BOUNDARYOUT_DATA	EM BOUNDARYOUT DATA entity code
Entity.EM_CIRCUIT0D_DATA	EM CIRCUIT0D DATA entity code
Entity.EM_CIRCUITRES_DATA	EM CIRCUITRES DATA entity code
Entity.EM_CIRCUIT_DATA	EM CIRCUIT DATA entity code
Entity.EM_GLOBAL_DATA	EM GLOBAL DATA entity code
Entity.EM_ISOPOTCONNOUT_DATA	EM ISOPOTCONNOUT DATA entity code
Entity.EM_ISOPOTOUT_DATA	EM ISOPOTOUT DATA entity code
Entity.EM_NODE_DATA	EM NODE DATA entity code
Entity.EM_PARTDATA_DATA	EM PARTDATA DATA entity code
Entity.EM_POINT_DATA	EM POINT DATA entity code
Entity.EM_RANDLESCELL_DATA	EM RANDLESCELL DATA entity code
Entity.EM_RISC_DATA	EM RANDLESINTSHORTCELL DATA entity code
Entity.EM_ROGOCOIL_DATA	EM ROGOCOIL DATA entity code
Entity.FSI	FSI entity code
Entity.FSI_SENSOR_DATA	FSI SENSOR DATA entity code
Entity.FSI_SURFACE_DATA	FSI SURFACE DATA entity code
Entity.GEOMETRIC_CONTACT	GEOMETRIC CONTACT entity code
Entity.ICFD	ICFD entity code
Entity.ICFD_DRAG_DATA	ICFD DRAG DATA entity code
Entity.ICFD_ICVOUT_ICVI_DATA	ICFD ICVOUT ICVI DATA entity code
Entity.ICFD_ICVOUT_ICV_DATA	ICFD ICVOUT ICV DATA entity code
Entity.ICFD_NODE_DATA	ICFD NODE DATA entity code
Entity.ICFD_POINT_DATA	ICFD POINT DATA entity code
Entity.ICFD_THERMAL_DATA	ICFD THERMAL DATA entity code
Entity.JOINT	JOINT entity code
Entity.JOINT_FLEXION_TORSION	FLEXION TORSION JOINT entity code
Entity.JOINT_GENERALIZED	GENERALIZED JOINT entity code
Entity.JOINT_JOINT	Conventional LS-DYNA JOINT entity code
Entity.JOINT_TRANSLATIONAL	TRANSLATIONAL JOINT entity code
Entity.MASS	MASS entity code
Entity.MODEL	MODEL entity code
Entity.NODAL_RB	NODAL RIGID BODY entity code

Entity.NODAL_RB_BODY	BODY in NODAL RIGID BODY entity code
Entity.NODAL_RB_PART	PART in NODAL RIGID BODY entity code
Entity.NODE	NODE entity code
Entity.NODE_GROUP	NODAL FORCE GROUP entity code
Entity.NODE_GROUP_GROUPS	GROUPS in NODAL FORCE GROUP entity code
Entity.NODE_GROUP_NODES	NODES in NODAL FORCE GROUP entity code
Entity.PART	PART entity code
Entity.PART_GROUP	PART GROUP entity code
Entity.PBLAST	PBLAST entity code
Entity.PBLAST_DATA	PBLAST DATA entity code
Entity.PBLAST_PART	PBLAST PART entity code
Entity.PRETENSIONER	PRETENSIONER entity code
Entity.PRTUBE	PRTUBE entity code
Entity.PULLEY	PULLEY entity code
Entity.RETRACTOR	RETRACTOR entity code
Entity.RIGIDWALL	RIGIDWALL entity code
Entity.SEATBELT	SEATBELT entity code
Entity.SHELL	SHELL entity code
Entity.SLIPRING	SLIPRING entity code
Entity.SOLID	SOLID entity code
Entity.SPC	SPC entity code
Entity.SPC_FORCES	SPC FORCES entity code
Entity.SPC_MODEL	SPC MODEL entity code
Entity.SPC_MOMENTS	SPC MOMENTS entity code
Entity.SPC_SET	SPC SET entity code
Entity.SPH	SPH entity code
Entity.SPRING	SPRING entity code
Entity.SPRING_ROTATIONAL	ROTATIONAL SPRING entity code
Entity.SPRING_TRANSLATIONAL	TRANSLATIONAL SPRING entity code
Entity.SUBSYSTEM	SUBSYSTEM entity code
Entity.THICK_SHELL	THICK SHELL entity code
Entity.TRACER	TRACER entity code
Entity.WELD	WELD entity code
Entity.WELD_ASSEMBLY	WELD ASSEMBLY entity code
Entity.WELD_CONSTRAINED	CONSTRAINED WELD entity code
Entity.WELD_GENERALISED	GENERALISED WELD entity code
Entity.WELD_NON_NODAL	NON-NODAL WELD entity code

Entity.WELD_SOLID	SOLID WELD entity code
Entity.WELD_SPOTWELD_BEAMS	SPOTWELD BEAMS entity code
Entity.X_SECTION	CROSS SECTION entity code

Detailed Description

The Entity class is used to define entity type codes that can then be compared with the entity Curve property and input for functions in the Model class.

```
Node = Entity.NODE;
```

File class

The File class allows you to read and write text files. More...

The T/HIS JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Copy(source*[string]*, dest*[string]*)
- Delete(filename*[string]*)
- DriveMapFilename(filename*[string]*, format*[constant]*)
- Exists(filename*[string]*)
- FindFiles(directory*[string]*, type (optional)*[constant]*)
- Get(url*[string]*, filename*[string]*, options (optional)*[object]*)
- IsAbsolute(filename*[string]*)
- IsDirectory(filename*[string]*)
- IsFile(filename*[string]*)
- IsReadable(filename*[string]*)
- IsWritable(filename*[string]*)
- Mkdir(directory*[string]*)
- Mktemp()
- Proxy(name*[string]*)
- ProxyPassword(name*[string]*)
- ProxyUsername(username*[string]*)
- ReadCSV(filename*[string]*, delimiter (optional)*[string]*, comment (optional)*[string]*)
- Rename(oldname*[string]*, newname*[string]*)
- Size(filename*[string]*)
- Upload(filename*[string]*, url*[string]*, options (optional)*[object]*)

Member functions

- Close()
- FindLineContaining(contain*[string]*)
- FindLineStarting(start*[string]*)
- Flush()
- ReadAll()
- ReadArrayBuffer(length (optional)*[integer]*)
- ReadChar()
- ReadLine()
- ReadLongLine()
- Seek(offset*[integer]*, origin (optional)*[constant]*)
- Tell()
- Write(string*[Any valid javascript type]*)
- WriteArrayBuffer(buffer*[ArrayBuffer]*, length (optional)*[integer]*)
- Writeln(string*[Any valid javascript type]*)

File constants

Name	Description
File.APPEND	Flag to open file for appending
File.BINARY	Flag to open file in binary mode. This will have no effect on unix/linux but for windows if a file is opened for writing with binary mode \n will not be translated to \r\n (CRLF), it will be written as \n (LF)
File.READ	Flag to open file for reading

File.UTF8	Flag to open file for reading as UTF-8 encoding.
File.WRITE	Flag to open file for writing

Constants for Find types

Name	Description
File.DIRECTORY	Find directories
File.FILE	Find files

Constants for Seek types

Name	Description
File.CURRENT	Seek relative to current file position
File.END	Seek relative to end of the file
File.START	Seek relative to start of the file

File properties

Name	Type	Description
filename (read only)	string	Name of the file
mode (read only)	constant	Mode the file was opened with (File.READ, File.WRITE etc)

Detailed Description

The File class gives you simple functions to read and write text files. The following simple example shows how to read from the file "/data/test/file.txt" and print each line read to the dialogue box:

```
var f, line;
f = new File("/data/test/file.txt", File.READ);
while ( (line = f.ReadLine()) != undefined)
{
    Message(line);
}
f.Close();
```

The following simple example shows how to write the numbers 1 to 10 to the file "/data/test/file.txt":

```
var n, line;
f = new File("/data/test/file.txt", File.WRITE);
for (n=1; n<=10; n++)
{
    f.Writeln(n);
}
f.Close();
```

See the documentation below for more details.

Constructor

new File(filename*[string]*, mode*[constant]*)

Description

Create a new File object for reading and writing text files.

Arguments

- **filename** (string)

Filename of the file you want to read/write. If reading, the file must exist. If writing, the file will be overwritten (if it exists) if mode is File.WRITE, or if mode is File.APPEND it will be appended to if it exists, or created if it does not. When reading a file the filename can also be a URL (uniform resource locator) in which case the file will be read from the remote site. See File.Get() for more details on the format of the URL.

- **mode** (constant)

The mode to open the file with. Can be File.READ, File.WRITE or File.APPEND. For File.WRITE or File.APPEND it can also be ORed with File.BINARY if required. By default text is read and written as ASCII. To read/write text in utf-8 mode can also be ORed with File.UTF8 if required.

Returns

File object

Return type

File

Example

To create a new file object to read file "/data/test/file.txt"

```
var f = new File("/data/test/file.txt", File.READ);
```

Details of functions

Close()

Description

Close a file opened by a File object.

Arguments

No arguments

Returns

No return value

Example

To close File object f.

```
f.Close();
```

Copy(source*[string]*, dest*[string]*) [static]

Description

Copies a file

Arguments

- **source** (string)

Source filename you want to copy.

- **dest** (string)

Destination filename you want to copy source file to.

Returns

true if copy successful, false otherwise.

Return type

Boolean

Example

To copy the file "/data/test/file.key" to "/data/test/file.key_backup"

```
var copied = File.Copy("/data/test/file.key", "/data/test/file.key_backup");
```

Delete(filename*[string]*) [static]

Description

Deletes a file

Arguments

- **filename** (string)

Filename you want to delete.

Returns

true if successful, false if not.

Return type

Boolean

Example

To delete the file "/data/test/file.key"

```
var deleted = File.Delete("/data/test/file.key");
```

DriveMapFilename(filename*[string]*, format*[constant]*) [static]

Description

Changes a filename or directory name to the correct format for a specific operating system using the directory mappings (if present)

Arguments

- **filename** (string)

Filename you want to drive map.

- **format** (constant)

The format for the file/directory name. Can be Include.NATIVE, Include.UNIX or Include.WINDOWS

Returns

string containing drive mapped filename

Return type

String

Example

If T/HIS has drive S: mapped to "/data" (by using the primer*drive_s, this*drive_s, d3plot*drive_s or oasys*drive_s preference)

```
var mapped = File.DriveMapFilename("/data/test/file.key", Include.WINDOWS);
```

mapped will be "S:\test\file.key".

```
var mapped = File.DriveMapFilename("S:\\test\\file.key", Include.UNIX);
```

mapped will be "/data/test/file.key".

Exists(filename*[string]*) [static]

Description

Check if a file exists. See also File.IsDirectory() and See also File.IsFile().

Arguments

- **filename** (string)

Filename you want to check for existance.

Returns

true/false

Return type

Boolean

Example

To see if the file "/data/test/file.key" exists

```
if (File.Exists("/data/test/file.key")) { do something }
```

FindFiles(directory*[string]*, type (optional)*[constant]*) [static]

Description

Find any files and/or directories in a directory.

Arguments

- **directory** (string)

Directory to look for files/directories in.

- **type (optional)** (constant)

Type of things to find. Can be bitwise OR of File.FILE and File.DIRECTORY. If omitted only files will be returned.

Returns

Array of filenames/directories

Return type

Array

Example

To return the filenames in the directory /data/test:

```
var fileList = File.FindFiles("/data/test")
```

To return the directories in the directory /data/test:

```
var fileList = File.FindFiles("/data/test", File.DIRECTORY)
```

To return the files and directories in the directory /data/test:

```
var fileList = File.FindFiles("/data/test", File.FILE|File.DIRECTORY)
```

FindLineContaining(contain*[string]*)

Description

Reads a line from a file which contains **contain**, opened for reading by a File object. Although this is possible using core JavaScript functions this function should be significantly faster as most of the processing is done by PRIMER in C rather than in the JavaScript interpreter. To enable this function to be as fast as possible a maximum line length of 512 characters is used. If you expect a file to have lines longer than 512 characters then use ReadLongLine which allows lines of any length. If one argument is used then the line must contain that string. If more than one argument is used then lines which contain any of the arguments will be returned

Arguments

- **contain** (string)

String which matching lines must contain

This argument can be repeated if required
Alternatively a single array argument containing the multiple values can be given

Returns

string read from file or `undefined` if end of file

Return type

String

Example

Loop, reading lines from File object f which contain 'example'.

```
var line;
while ( (line = f.FindLineContaining("example") ) != undefined)
{
}
```

FindLineStarting(start*[string]*)

Description

Reads a line from a file which starts with start, opened for reading by a File object. Although this is possible using core JavaScript functions this function should be significantly faster as most of the processing is done by PRIMER in C rather than in the JavaScript interpreter. To enable this function to be as fast as possible a maximum line length of 512 characters is used. If you expect a file to have lines longer than 512 characters then use ReadLongLine which allows lines of any length. If one argument is used then the line must start with that string. If more than one argument is used then lines which start with any of the arguments will be returned

Arguments

- **start** (string)

String which matching lines must start with

This argument can be repeated if required
Alternatively a single array argument containing the multiple values can be given

Returns

string read from file or `undefined` if end of file

Return type

String

Example

Loop, reading lines from File object f which start 'example'.

```
var line;
while ( (line = f.FindLineStarting("example") ) != undefined)
{
}
```

Flush()

Description

Flushes a file opened for writing by a File object.

Arguments

No arguments

Returns

No return value

Example

To flush File object f.

```
f.Flush();
```

Get(url*[string]*, filename*[string]*, options (optional)*[object]*) [static]

Description

Get a file from a remote location. See also File.Proxy(), File.ProxyPassword() and File.ProxyUsername().

Arguments

- **url** (string)

URL (uniform resource locator) of remote file you want to get. Currently http and ftp are supported. For http give the full address including the leading 'http://'. e.g.
'http://www.example.com/file.html'.
For ftp an optional username and password can be given. e.g.
'ftp://ftp.example.com' retrieves the directory listing for the root directory.
'ftp://ftp.example.com/readme.txt' downloads the file readme.txt from the root directory.
'ftp://user:password@ftp.example.com/readme.txt' retrieves the readme.txt file from the user's home directory.

- **filename** (string)

Filename you want to save the file to.

- **options (optional)** (object)

Options for get. If 'username' and 'password' are set then basic authorization using the username and password will be used.

Object has the following properties:

Name	Type	Description

password (optional)	string	Password
response (optional)	boolean	If set to true, then the response code will be returned instead of true/false. This can be used to retieve error messages and codes when the file is not returned successfully.
username (optional)	string	Username

Returns

true if file was successfully got, false otherwise.

Return type

Boolean

Example

To get the file "http://www.example.com/file.html" and save it to C:\temp:

```
File.Get("http://www.example.com/file.html", "C:\temp\file.html");
```

IsAbsolute(filename*[string]*) [static]

Description

Check if a filename is absolute or relative.

Arguments

- **filename** (string)

Filename you want to check.

Returns

true/false

Return type

Boolean

Example

To see if the filename "/data/test" is absolute (which it is!)

```
if (File.IsAbsolute("/data/test")) { do something }
```

IsDirectory(filename*[string]*) [static]

Description

Check if a filename is a directory. See also File.Exists(), File.IsFile(), File.IsReadable() and File.IsWritable().

Arguments

- **filename** (string)

Filename you want to check.

Returns

true/false

Return type

Boolean

Example

To see if the filename "/data/test" is a directory

```
if (File.IsDirectory("/data/test")) { do something }
```

IsFile(filename*[string]*) [static]

Description

Check if a filename is a file. See also File.Exists(), File.IsDirectory(), File.IsReadable() and File.IsWritable().

Arguments

- **filename** (string)

Filename you want to check.

Returns

true/false

Return type

Boolean

Example

To see if the filename "/data/test" is a file

```
if (File.IsFile("/data/test")) { do something }
```

IsReadable(filename*[string]*) [static]

Description

Check if a filename has read permissions. See also File.Exists(), File.IsDirectory() and File.IsWritable().

Arguments

- **filename** (string)

Filename you want to check.

Returns

true/false

Return type

Boolean

Example

To see if the filename "/data/test" is readable

```
if (File.IsReadable("/data/test")) { do something }
```

IsWritable(filename*[string]*) [static]

Description

Check if a filename has write permissions. If *filename* exists and it is a file then it is checked to see if it can be opened with write (File.APPEND permissions). If *filename* exists and it is a directory then the directory is checked for write permission (can files be created in the directory). If *filename* does not exist then it is assumed to be a file and is checked to see if it can be opened for writing (File.WRITE permissions). See also File.Exists(), File.IsDirectory() and File.IsReadable().

Arguments

- **filename** (string)

Filename you want to check.

Returns

true/false

Return type

Boolean

Example

To see if the filename "/data/test" is writable

```
if (File.IsWritable("/data/test")) { do something }
```

Mkdir(directory*[string]*) [static]

Description

Make a directory. If PRIMER preference 'directory_permission' is set e.g.755 then this will apply (same as if set by chmod 755) ignoring any setting of umask. If there is no preference then the users current setting of umask will control permissions (same as system mkdir)

Arguments

- **directory** (string)

The name of the directory you want to create.

Returns

true if successfully created, false if not.

Return type

Boolean

Example

To make the directory "/data/test"

```
var success = File.Mkdir("/data/test");
```

Mktemp() [static]

Description

Make a temporary filename for writing a temporary file.

Arguments

No arguments

Returns

String name of temporary filename that can be used.

Return type

String

Example

To get a temp filename"

```
var filename = File.Mktemp();
```

Proxy(name*[string]*) [static]

Description

Set a proxy for files opened by http, ftp etc. See also File.Get(), File.ProxyPassword() and File.ProxyUsername().

Arguments

- **name** (string)

The name of the proxy.

Returns

No return value

Example

To set the proxy to "http://example.proxy.com" using port 80:

```
File.Proxy("http://example.proxy.com:80");
```

ProxyPassword(name*[string]*) [static]

Description

Set a proxy password for files opened by http, ftp etc. See also File.Get(), File.Proxy() and File.ProxyUsername().

Arguments

- **name** (string)

Password for the proxy server.

Returns

No return value

Example

To set the proxy password to "password":

```
File.ProxyPassword("password");
```

ProxyUsername(username*[string]*) [static]

Description

Set a proxy username for files opened by http, ftp etc. See also File.Get(), File.Proxy() and File.ProxyPassword().

Arguments

- **username** (string)

The username for the proxy.

Returns

No return value

Example

To set the proxy username to "username":

```
File.ProxyUsername("username");
```

ReadAll()

Description

Reads **all** the remaining characters from a file opened for reading by a File object. As this function can read the entire file as a string be careful when reading large files as it will consume large amounts of memory.

Arguments

No arguments

Returns

String. Characters read from file or undefined if end of file

Return type

String

Example

Read all characters from File object f.

```
var c = f.ReadAll();
```

ReadArrayBuffer(length (optional)*[integer]*)

Description

Reads binary data from a file opened for reading by a File object. The data is returned as an ArrayBuffer object. For more details on how to use an ArrayBuffer see the following links:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Typed_arrays
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/ArrayBuffer
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/TypedArray
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/DataView.

Arguments

- **length (optional)** (integer)

Number of bytes to try to read from the file. If omitted all the remaining data from the file will be read.

Returns

ArrayBuffer object or undefined if end of file

Return type

ArrayBuffer

Example

To read data as 32bit unsigned integers from File object f.

```
var ab = f.ReadArrayBuffer();
var u32 = new Uint32Array(ab);
for (var i=0; i<u32.length; i++
{
    var value = u32[i];
}
```

ReadCSV(filename*[string]*, delimiter (optional)*[string]*, comment (optional)*[string]*) [static]

Description

Reads the input CSV file and returns an array of string arrays. If the CSV file has legitimate records the function returns an Array object containing sub-arrays of strings otherwise the function returns NULL. The lengths of all the sub-arrays are the same and equal to maximum number of fields in any of the records. For records in a CSV file having fewer fields, the respective sub-arrays are padded with NULL elements to the maximum array length.

Arguments

- **filename** (string)

Filename you want to read CSV options from.

- **delimiter (optional)** (string)

Delimiter string to be used. Default is a comma (",").

- **comment (optional)** (string)

Comment string to be used. Default is a dollar sign ("$").

Returns

2d array of strings.

Return type

String

Example

To Read CSV file "sample.csv" and print all records to a Window.

```
var csv_file_path = "C:\\sample.csv";
var records = "";
if(!File.Exists(csv_file_path))
{
    Window.Information("CSV file %s not present", csv_file_path);
    Exit();
}
var csv_array = File.ReadCSV(csv_file_path);
if(csv_array != null)
{
    for(var i = 0; i < csv_array.length; i++)
    {
        var record_array = csv_array[i];
        for(var j = 0; j < record_array.length; j++)
        {
            if(record_array[j] != null)
                records = records + record_array[j] + " , ";
        }
        records = records + "\n";
    }
}
Options.max_window_lines = csv_array.length;
Window.Information("File.ReadCSV Ouptut", records);
```

To Read CSV file "sample.csv" with delimiter string "::" and comment string "##".

```
var csv_array = File.ReadCSV(csv_file_path, "::", "##");
```

ReadChar()

Description

Reads a single character from a file opened for reading by a File object.

Arguments

No arguments

Returns

character read from file or

`undefined`

if end of file

Return type

String

Example

Loop, reading characters from File object f.

```
var c;
while ( (c = f.ReadChar()) != undefined) { ... }
```

ReadLine()

Description

Reads a line from a file opened for reading by a File object. To enable this function to be as fast as possible a maximum line length of 512 characters is used. If you expect a file to have lines longer than 512 characters then use ReadLongLine which allows lines of any length.

Arguments

No arguments

Returns

string read from file or

`undefined`

if end of file

Return type

String

Example

Loop, reading lines from File object f.

```
var line;
while ( (line = f.ReadLine()) != undefined) { ... }
```

ReadLongLine()

Description

Reads a line from a file opened for reading by a File object. The line can be any length. If your file has lines shorter than 512 characters then you may want to use ReadLine instead which is faster.

Arguments

No arguments

Returns

string read from file or

`undefined`

if end of file

Return type

String

Example

Loop, reading lines from File object f.

```
var line;
while ( (line = f.ReadLongLine()) != undefined) { ... }
```

Rename(oldname*[string]*, newname*[string]*) [static]

Description

Rename an existing file to have a different name.

Arguments

- **oldname** (string)

Existing filename you want to rename

- **newname** (string)

New filename you want to rename to

Returns

true if successful, false if not.

Return type

Boolean

Example

To rename the file "/data/test/file.key" to "/data/test/new_file.key"

```
var size = File.Rename("/data/test/file.key", "/data/test/new_file.key");
```

Seek(offset*[integer]*, origin (optional)*[constant]*)

Description

Set the current position for reading or writing in a File object.

Arguments

- **offset** (integer)

Offset to seek to in the file

- **origin (optional)** (constant)

Origin for offset. Must be one of File.START, File.END or File.CURRENT. If omitted File.START will be used.

Returns

no return value

Example

To seek to the end of File f:

```
f.Seek(0, File.END);
```

To seek to the beginning of File f:

```
f.Seek(0, File.START);
```

To move forward 10 characters in File f:

```
f.Seek(10, File.CURRENT);
```

Size(filename*[string]*) [static]

Description

Return the size of a file in bytes

Arguments

- **filename** (string)

Filename you want the size of.

Returns

size in bytes

Return type

Number

Example

To get the size of the file "/data/test/file.key"

```
var size = File.Size("/data/test/file.key");
```

Tell()

Description

Return the current file position for a File object. Note that on Windows when reading files if the file is not opened with File.BINARY this may not return the correct file position for files with unix line endings.

Arguments

No arguments

Returns

integer

Return type

Number

Example

To get the current file position for File f:

```
var pos = f.Tell();
```

Upload(filename*[string]*, url*[string]*, options (optional)*[object]*) [static]

Description

Uploads a file to a remote location. See also File.Proxy(), File.ProxyPassword() and File.ProxyUsername().

Arguments

- **filename** (string)

Filename you want to upload.

- **url** (string)

URL (uniform resource locator) of the remote location you want to upload the file to. Currently only http is supported. Give the full address including the leading 'http://'. e.g. 'http://www.example.com/file.html'.

- **options (optional)** (object)

Options for upload. If both of these are set then basic authorization using the username and password will be used.

Object has the following properties:

Name	Type	Description
password (optional)	string	Password
username (optional)	string	Username

Returns

true if file was successfully uploaded, false otherwise.

Return type

Boolean

Example

To upload the file "C:\temp\file.txt" to "http://www.example.com/file.txt":

```
File.Upload("C:/temp/file.txt", "http://www.example.com/file.txt");
```

Write(string*[Any valid javascript type]*)

Description

Write a string to a file opened for writing by a File object. **Note that a carriage return is not added**.

Arguments

- **string** (Any valid javascript type)

The string/item that you want to write

Returns

No return value

Example

To write string "Hello, world!" to File object f

```
f.Write("Hello, world!\n");
```

To write the title of model m to File object f

```
f.Write("The title of model 2 is " + m.title + "\n");
```

WriteArrayBuffer(buffer*[ArrayBuffer]*, length (optional)*[integer]*)

Description

Writes binary data to a file opened for writing by a File object. The data to write is an ArrayBuffer object. For more details on how to use an ArrayBuffer see the following links:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Typed_arrays
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/ArrayBuffer
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/TypedArray
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/DataView.

Arguments

- **buffer** (ArrayBuffer)

ArrayBuffer to write to file

- **length (optional)** (integer)

Number of bytes to write to the file. If omitted all the data in the ArrayBuffer will be written (buffer.byteLength bytes)

Returns

No return value

Example

To write ArrayBuffer ab to File object f.

```
f.WriteArrayBuffer(ab);
```

Writeln(string*[Any valid javascript type]*)

Description

Write a string to a file opened for writing by a File object **adding a carriage return**.

Arguments

- **string** (Any valid javascript type)

The string/item that you want to write

Returns

No return value

Example

To write string "Hello, world!" to File object f automatically adding a carriage return

```
f.Writeln("Hello, world!");
```

To write the title of model m to File object f automatically adding a carriage return

```
f.Writeln("The title of model 2 is " + m.title);
```

Graph class

The Graph class gives you access to graphs in T/HIS. More...

The T/HIS JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- DeleteFromID(ID*[integer]*)
- GetFromID(ID*[integer]*)
- MaxAllowable()
- Total()

Member functions

- AddCurveID(Curve ID*[integer]*, redraw (optional)*[boolean]*)
- AddCurveID(Curve ID*[integer]*, No redraw (optional)*[integer]*) [deprecated]
- AddToPage(Page number*[integer]*)
- Delete()
- GetAllCurveIDs()
- GetAllPageIDs()
- GetNumCurves()
- Lock(Lock type*[integer]*)
- RemoveCurveID(ID*[integer]*)
- RemoveFromPage(ID*[integer]*)

Graph constants

Name	Description
Graph.AXIS_LINEAR	Linear axis type
Graph.AXIS_LOG	Logarithmic axis type
Graph.AXIS_UNITS_AUTO	Auto format axis units
Graph.AXIS_UNITS_GENERAL	General format for axis units
Graph.AXIS_UNITS_SCIENTIFIC	Scientific format for axis units
Graph.FONT_COURIER_BOLD	Courier bold font
Graph.FONT_COURIER_ MEDIUM	Courier medium font
Graph.FONT_DEFAULT	Takes the font defined in the preference file
Graph.FONT_HELVETICA_ BOLD	Helvetical bold font
Graph.FONT_HELVETICA_ MEDIUM	Helvetical medium font
Graph.FONT_SIZE_10	10 point font size
Graph.FONT_SIZE_12	12 point font size

Graph.FONT_SIZE_14	14 point font size
Graph.FONT_SIZE_18	18 point font size
Graph.FONT_SIZE_24	24 point font size
Graph.FONT_SIZE_8	8 point font size
Graph.FONT_SIZE_AUTO	Font size would be automatically adjusted based on the graph area
Graph.FONT_TIMES_BOLD	Times New Roman bold font
Graph.FONT_TIMES_MEDIUM	Times New Roman medium font
Graph.GRID_OFF	Turn off the grid.
Graph.GRID_ON	Turn on the grid.
Graph.LEGEND_1_COLUMN	Curve labels will be displayed in a single column in the legend
Graph.LEGEND_2_COLUMN	Curve labels will be displayed in two columns in the legend
Graph.LEGEND_3_COLUMN	Curve labels will be displayed in three columns in the legend
Graph.LEGEND_AUTO	Automatic legend layout (see Legend)
Graph.LEGEND_COLUMN_LIST	Column list legend layout (see Legend)
Graph.LEGEND_FLOATING	Floating legend layout (see Legend)
Graph.LEGEND_OFF	Off legend layout (see Legend)
Graph.NO	Flag for no.
Graph.OFF	Flag to turn off.
Graph.ON	Flag to turn on.
Graph.PREFIX_AUTO	Automatically add prefix to the curve label in the legend (see Legend)
Graph.PREFIX_DIR	Directory name of the model will be used as the curve label prefix in the legend (see Legend)
Graph.PREFIX_MODEL_ NUMBER	Model number will be used as the curve label prefix in the legend (see Legend)
Graph.PREFIX_OFF	Turn off the curve label prefix in the legend (see Legend)
Graph.PREFIX_ON	Add prefix to the curve label in the legend (see Legend)
Graph.PREFIX_THF	Root name of the THF file will be used as the curve label prefix in the legend (see Legend)
Graph.PREFIX_USER_DEFINED	A user defined prefix will be used as the curve label prefix in the legend (see Legend)
Graph.YES	Flag for yes.

Graph properties

Name	Type	Description
active	constant	If the graph is active or inactive. Can take Graph.YES or Graph.NO
add_x_units	constant	shows x-axis units. It can take either Graph.ON or Graph.OFF
add_y2_units	constant	shows second y-axis units. It can take either Graph.ON or Graph.OFF
add_y_units	constant	shows y-axis units. It can take either Graph.ON or Graph.OFF

auto_title	string	Turn on to set graph title automatically and turn off to define the graph title manually using the property Graph.title. Can take either Graph.ON or Graph.OFF
auto_xlabel	constant	Turn on to set label for the x-axis automatically and turn off to define the label for the x-axis manually using the property xlabel. Can take either Graph.ON or Graph.OFF
auto_xmax	constant	Can take either Graph.ON or Graph.OFF. Graph.ON will set the maximum value for the y-axis range automatically and Graph.OFF will use the property xmax value as the maximum value for the x-axis range
auto_xmin	constant	Can take either Graph.ON or Graph.OFF. Graph.ON will set the minimum value for the x-axis range automatically and Graph.OFF will use the property xmin value as the minimum value for the x-axis range
auto_y2label	constant	Turn on to set label for the second y-axis automatically and turn off to define the label for the second y-axis manually using the property y2label. Can take either Graph.ON or Graph.OFF
auto_y2max	constant	Can take either Graph.ON or Graph.OFF. Graph.ON will set the maximum value for the second y-axis range automatically and Graph.OFF will use the property y2max value as the maximum value for the second y-axis range
auto_y2min	constant	Can take either Graph.ON or Graph.OFF. Graph.ON will set the minimum value for the second y-axis range automatically and Graph.OFF will use the property y2min value as the minimum value for the second y-axis range
auto_ylabel	constant	Turn on to set label for the y-axis automatically and turn off to define the label for the y-axis manually using the property ylabel. Can take either Graph.ON or Graph.OFF
auto_ymax	constant	Can take either Graph.ON or Graph.OFF. Graph.ON will set the maximum value for the y-axis range automatically and Graph.OFF will use the property ymax value as the maximum value for the y-axis range
auto_ymin	constant	Can take either Graph.ON or Graph.OFF. Graph.ON will set the minimum value for the y-axis range automatically and Graph.OFF will use the property ymin value as the minimum value for the y-axis range
background_ colour	Colour	Graph background colour
foreground_ colour	Colour	Graph foreground colour
grid	constant	To turn on/off the grid. Can take Graph.GRID_ON or Graph.GRID_OFF
id (read only)	integer	Graph ID
legend_ background_ colour	Colour	Background colour for the legend area
legend_ background_ trans	integer	Transparancy of the legend area. The value should lie between 0 and 100
legend_font	constant	Font for the curve labels in the legend. Can take either Graph.FONT_DEFAULT, Graph.FONT_HELVETICA_MEDIUM, Graph.FONT_HELVETICA_BOLD, Graph.FONT_TIMES_MEDIUM, Graph.FONT_TIMES_BOLD, Graph.FONT_ COURIER_MEDIUM or Graph.FONT_COURIER_BOLD
legend_font_ colour	Colour	Font colour for the curve labels in the legend
legend_font_ size	constant	Font size for the curve labels in the legend. Can take either Graph.FONT_SIZE_AUTO, Graph.FONT_SIZE_8, Graph.FONT_SIZE_10, Graph.FONT_SIZE_12, Graph.FONT_ SIZE_14, Graph.FONT_SIZE_18 or Graph.FONT_SIZE_24
legend_layout	constant	Defines the legend layout type. Can take Graph.LEGEND_COLUMN_LIST, Graph.LEGEND_AUTO, Graph.LEGEND_OFF or Graph.LEGEND_FLOATING

legend_prefix_format	constant	Format of the prefix that is being included in the curve label of the legend. Can take either Graph.PREFIX_MODEL_NUMBER, Graph.DIR, Graph.PREFIX_THF or Graph.PREFIX_USER_DEFINED
legend_show_prefix	constant	Include the prefix in the curve label of the legend. Can take either Graph.PREFIX_AUTO, Graph.PREFIX_ON or Graph.PREFIX_OFF
legend_show_user_lines	constant	Visibility of user lines when Graph.LEGEND_COLUMN_LIST is selected for legend layout. Can take either Graph.ON or Graph.OFF
legend_user_line_1	string	User defined line 1 from the legend area
legend_user_line_1_size	constant	Font size for the user defined line 1. Can take either Graph.FONT_SIZE_AUTO, Graph.FONT_SIZE_8, Graph.FONT_SIZE_10, Graph.FONT_SIZE_12, Graph.FONT_SIZE_14, Graph.FONT_SIZE_18 or Graph.FONT_SIZE_24
legend_user_line_2	string	User defined line 2 from the legend area
legend_user_line_2_size	constant	Font size for the user defined line 2. Can take either Graph.FONT_SIZE_AUTO, Graph.FONT_SIZE_8, Graph.FONT_SIZE_10, Graph.FONT_SIZE_12, Graph.FONT_SIZE_14, Graph.FONT_SIZE_18 or Graph.FONT_SIZE_24
legend_user_line_3	string	User defined line 3 from the legend area
legend_user_line_3_size	constant	Font size for the user defined line 3. Can take either Graph.FONT_SIZE_AUTO, Graph.FONT_SIZE_8, Graph.FONT_SIZE_10, Graph.FONT_SIZE_12, Graph.FONT_SIZE_14, Graph.FONT_SIZE_18 or Graph.FONT_SIZE_24
legend_user_line_4	string	User defined line 4 from the legend area
legend_user_line_4_size	constant	Font size for the user defined line 4. Can take either Graph.FONT_SIZE_AUTO, Graph.FONT_SIZE_8, Graph.FONT_SIZE_10, Graph.FONT_SIZE_12, Graph.FONT_SIZE_14, Graph.FONT_SIZE_18 or Graph.FONT_SIZE_24
legend_user_line_5	string	User defined line 6 from the legend area
legend_user_line_5_size	constant	Font size for the user defined line 5. Can take either Graph.FONT_SIZE_AUTO, Graph.FONT_SIZE_8, Graph.FONT_SIZE_10, Graph.FONT_SIZE_12, Graph.FONT_SIZE_14, Graph.FONT_SIZE_18 or Graph.FONT_SIZE_24
legend_user_line_6	string	User defined line 6 from the legend area
legend_user_line_6_size	constant	Font size for the user defined line 6. Can take either Graph.FONT_SIZE_AUTO, Graph.FONT_SIZE_8, Graph.FONT_SIZE_10, Graph.FONT_SIZE_12, Graph.FONT_SIZE_14, Graph.FONT_SIZE_18 or Graph.FONT_SIZE_24
legend_user_lines_colour	Colour	Font colour for the user defined lines in the legend
legend_user_lines_font	constant	Font for the user defined lines in the legend. Can take either Graph.FONT_DEFAULT, Graph.FONT_HELVETICA_MEDIUM, Graph.FONT_HELVETICA_BOLD, Graph.FONT_TIMES_MEDIUM, Graph.FONT_TIMES_BOLD, Graph.FONT_COURIER_MEDIUM or Graph.FONT_COURIER_BOLD
num_legend_columns	constant	Number of columns of curve labels in legends. Can take Graph.LEGEND_1_COLUMN, Graph.LEGEND_2_COLUMN or Graph.LEGEND_3_COLUMN
show_title	string	Shows graph title. Can take either Graph.ON or Graph.OFF
show_xlabel	constant	Shows graph x-axis label. Can take either Graph.ON or Graph.OFF

show_y2axis	constant	Shows graph second y-axis. Can take either Graph.ON or Graph.OFF. (Note, Curve property y_axis must be set to Curve.Y2_AXIS for the curve to be plotted on the second y-axis.)
show_y2label	constant	Shows graph second y-axis label. Can take either Graph.ON or Graph.OFF
show_ylabel	constant	Shows graph y-axis label. Can take either Graph.ON or Graph.OFF
title	string	Graph title
x_axis_type	constant	Defines x-axis type i.e. linear or logarithmic. Can take either Graph.AXIS_LINEAR or Graph.AXIS_LOG
x_unit_colour	Colour	Colour of the x-axis units
x_unit_decimals	integer	Defines the number decimals in the x-axis units.
x_unit_font	constant	Font for the x-axis units. Can take either Graph.FONT_DEFAULT, Graph.FONT_HELVETICA_MEDIUM, Graph.FONT_HELVETICA_BOLD, Graph.FONT_TIMES_MEDIUM, Graph.FONT_TIMES_BOLD, Graph.FONT_COURIER_MEDIUM or Graph.FONT_COURIER_BOLD
x_unit_format	constant	Defines the format for the x-axis units. Can take either Graph.AXIS_UNITS_AUTO, Graph.AXIS_UNITS_SCIENTIFIC or Graph.AXIS_UNITS_GENERAL
x_unit_size	constant	Font size for the x-axis units. Can take either Graph.FONT_SIZE_AUTO, Graph.FONT_SIZE_8, Graph.FONT_SIZE_10, Graph.FONT_SIZE_12, Graph.FONT_SIZE_14, Graph.FONT_SIZE_18 or Graph.FONT_SIZE_24
xlabel	string	Label for x-axis
xlabel_colour	Colour	Colour of the x-axis label
xlabel_font	constant	Font for the x-axis label. Can take either Graph.FONT_DEFAULT, Graph.FONT_HELVETICA_MEDIUM, Graph.FONT_HELVETICA_BOLD, Graph.FONT_TIMES_MEDIUM, Graph.FONT_TIMES_BOLD, Graph.FONT_COURIER_MEDIUM or Graph.FONT_COURIER_BOLD
xlabel_size	constant	Font size for the x-axis label. Can take either Graph.FONT_SIZE_AUTO, Graph.FONT_SIZE_8, Graph.FONT_SIZE_10, Graph.FONT_SIZE_12, Graph.FONT_SIZE_14, Graph.FONT_SIZE_18 or Graph.FONT_SIZE_24
xmax	real	Maximum value of x-axis range
xmin	real	Minimum value of the x-axis range
y2_axis_type	constant	Defines second y-axis type i.e. linear or logarithmic. Can take either Graph.AXIS_LINEAR or Graph.AXIS_LOG
y2_unit_colour	Colour	Colour of the second y-axis units
y2_unit_decimals	integer	Defines the number decimals in the second y-axis units.
y2_unit_font	constant	Font for the second y-axis label. Can take either Graph.FONT_DEFAULT, Graph.FONT_HELVETICA_MEDIUM, Graph.FONT_HELVETICA_BOLD, Graph.FONT_TIMES_MEDIUM, Graph.FONT_TIMES_BOLD, Graph.FONT_COURIER_MEDIUM or Graph.FONT_COURIER_BOLD
y2_unit_format	constant	Defines the format for the second y-axis units. Can take either Graph.AXIS_UNITS_AUTO, Graph.AXIS_UNITS_SCIENTIFIC or Graph.AXIS_UNITS_GENERAL
y2_unit_size	constant	Font size for the second y-axis units. Can take either Graph.FONT_SIZE_AUTO, Graph.FONT_SIZE_8, Graph.FONT_SIZE_10, Graph.FONT_SIZE_12, Graph.FONT_SIZE_14, Graph.FONT_SIZE_18 or Graph.FONT_SIZE_24
y2label	string	Label for second y-axis

y2label_colour	Colour	Colour of the second y-axis label
y2label_font	constant	Font for the second y-axis label. Can take either Graph.FONT_DEFAULT, Graph.FONT_HELVETICA_MEDIUM, Graph.FONT_HELVETICA_BOLD, Graph.FONT_TIMES_MEDIUM, Graph.FONT_TIMES_BOLD, Graph.FONT_COURIER_MEDIUM or Graph.FONT_COURIER_BOLD
y2label_size	constant	Font size for the second y-axis label. Can take either Graph.FONT_SIZE_AUTO, Graph.FONT_SIZE_8, Graph.FONT_SIZE_10, Graph.FONT_SIZE_12, Graph.FONT_SIZE_14, Graph.FONT_SIZE_18 or Graph.FONT_SIZE_24
y2max	real	Maximum value of the second y-axis range
y2min	real	Minimum value of the second y-axis range
y_axis_type	constant	Defines y-axis type i.e. linear or logarithmic. Can take either Graph.AXIS_LINEAR or Graph.AXIS_LOG
y_unit_colour	Colour	Colour of the y-axis units
y_unit_decimals	integer	The number decimals in the y-axis units.
y_unit_font	constant	Font for the y-axis units. Can take either Graph.FONT_DEFAULT, Graph.FONT_HELVETICA_MEDIUM, Graph.FONT_HELVETICA_BOLD, Graph.FONT_TIMES_MEDIUM, Graph.FONT_TIMES_BOLD, Graph.FONT_COURIER_MEDIUM or Graph.FONT_COURIER_BOLD
y_unit_format	constant	Defines the format for the y-axis units. Can take either Graph.AXIS_UNITS_AUTO, Graph.AXIS_UNITS_SCIENTIFIC or Graph.AXIS_UNITS_GENERAL
y_unit_size	constant	Font size for the y-axis units. Can take either Graph.FONT_SIZE_AUTO, Graph.FONT_SIZE_8, Graph.FONT_SIZE_10, Graph.FONT_SIZE_12, Graph.FONT_SIZE_14, Graph.FONT_SIZE_18 or Graph.FONT_SIZE_24
ylabel	string	Label for y-axis
ylabel_colour	Colour	Colour of the y-axis label
ylabel_font	constant	Font for the y-axis label. Can take either Graph.FONT_DEFAULT, Graph.FONT_HELVETICA_MEDIUM, Graph.FONT_HELVETICA_BOLD, Graph.FONT_TIMES_MEDIUM, Graph.FONT_TIMES_BOLD, Graph.FONT_COURIER_MEDIUM or Graph.FONT_COURIER_BOLD
ylabel_size	constant	Font size for the y-axis label. Can take either Graph.FONT_SIZE_AUTO, Graph.FONT_SIZE_8, Graph.FONT_SIZE_10, Graph.FONT_SIZE_12, Graph.FONT_SIZE_14, Graph.FONT_SIZE_18 or Graph.FONT_SIZE_24
ymax	real	Maximum value of y-axis range
ymin	real	Minimum value of the y-axis range

Detailed Description

The Graph class contains information on the number of graphs. See the documentation below for more details.

Constructor

new Graph(index (optional)*[integer]*)

Description

Create a new Graph.

Arguments

- **index (optional)** (integer)

Graph index to copy initial display and axis settings from (optional). If not defined then the display and axis settings will be copied from those defined in the preference file.

Returns

Graph object

Return type

Graph

Example

To create a new graph and copy all of the setting from graph 2

```
var l = new Graph(2);
```

Details of functions

AddCurveID(Curve ID*[integer]*, redraw (optional)*[boolean]*)

Description

Adds a curve to the graph.

Arguments

- **Curve ID** (integer)

ID of the curve to add.

- **redraw (optional)** (boolean)

If this argument is false then the graph will not be redrawn after the curve is added. This is to be used if a large number of curves are to be added to a graph, so as to avoid the same curves being drawn multiple times. No argument or true will trigger a redraw after the curve is added.

Returns

Returns true if the curve is successfully added to the graph else it would return false

Return type

Boolean

Example

To add a curve with id (n) to the graph (g):

```
g.AddCurveID(n);
```

AddCurveID(Curve ID*[integer]*, No redraw (optional)*[integer]*) [deprecated]

This function is deprecated in version 22.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Adds a curve to the graph.

Arguments

- **Curve ID** (integer)

ID of the curve to add.

- **No redraw (optional)** (integer)

If this argument is 1 then the graph will not be redrawn after the curve is added. This is to be used if a large number of

curves are to be added to a graph, so as to avoid the same curves being drawn multiple times. No argument or 0 will trigger a redraw after the curve is added.

Returns

Returns true if the curve is successfully added to the graph else it would return false

Return type

Boolean

Example

To add a curve with id (n) to the graph (g):

```
g.AddCurveID(n);
```

AddToPage(Page number*[integer]*)

Description

Adds the graph to the page.

Arguments

- **Page number** (integer)

Page number for which to add the graph to.

Returns

Returns true if the graph is successfully added to the page else it would return false

Return type

Boolean

Example

To add a graph (g) to page id (n):

```
g.AddToPage(n);
```

Delete()

Description

Deletes the graph

Arguments

No arguments

Returns

No return value

Example

Deletes the graph (g)

```
g.Delete();
```

DeleteFromID(ID*[integer]*) [static]

Description

Deletes a graph

Arguments

- **ID** (integer)

ID of graph to delete

Returns

No return value

Example

To delete the graph n

```
Graph.DeleteFromID(n);
```

GetAllCurveIDs()

Description

Returns the IDs of the curves present in the graph in an array.

Arguments

No arguments

Returns

An array of numbers containing the curve IDs

Return type

Number

Example

To get the array of all the curve ids present in a graph (g):

```
var num = g.GetAllCurveIDs();
```

GetAllPageIDs()

Description

Returns all the pages containing the graph.

Arguments

No arguments

Returns

An array of numbers containing the page IDs

Return type

Number

Example

To get the list of all page ids containing the graph (g):

```
var pages_ids = g.GetAllPageIDs();
```

GetFromID(ID*[integer]*) [static]

Description

Returns the graph object for a given graph id.

Arguments

- **ID** (integer)

ID of graph to return the graph for

Returns

Graph object or NULL if graph does not exists

Return type

Graph

Example

To get the graph n

```
var num = Graph.GetFromID(n);
```

GetNumCurves()

Description

Returns number curves present in the graph.

Arguments

No arguments

Returns

Number of curves present in the graph.

Return type

Number

Example

To find number of curves in a graph (g):

```
var num = g.GetNumCurves();
```

Lock(Lock type*[integer]*)

Description

Locks the blanking status of either blanked curves, unblanked curves or all curves on the graph.

Arguments

- **Lock type** (integer)

No argument or 0 to lock blanked curves, -1 to unlock blanked curves, -2 to unfreeze all visible curves

Returns

No return value

Example

To lock all blanked curves on graph g:

```
g.Lock();
```

MaxAllowable() [static]

Description

Returns the maximum allowable number of graphs. Useful because this differs between linked and unlinked T/HIS and can be used to check before trying to reference a graph number that may be out-of-range.

Arguments

No arguments

Returns

integer

Return type

Number

Example

To find the maximum allowable number of graphs in T/HIS:

```
var maxGraphs = Graph.MaxAllowable();
```

RemoveCurveID(ID*[integer]*)

Description

Removes a curve from the graph.

Arguments

- **ID** (integer)

ID of the curve to be removed

Returns

Returns true if the curve is successfully removed from the graph else it would return false

Return type

Boolean

Example

To remove a curve with id (n) from the graph (g):

```
g.RemoveCurveID(n);
```

RemoveFromPage(ID*[integer]*)

Description

Removes the graph from a page.

Arguments

- **ID** (integer)

ID of the page from which the graph is to be removed

Returns

Returns true if the graph is successfully removed from the page else it would return false

Return type

Boolean

Example

To remove the graph (g) from page with id (n):

```
g.RemoveFromPage(n);
```

Total() [static]

Description

Returns the total number of graphs.

Arguments

No arguments

Returns

integer

Return type

Number

Example

To find how many graphs there are in T/HIS:

```
var num = Graph.Total();
```

Group class

The Group class gives you access to groups in T/HIS. More...

The T/HIS JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- DeleteGroup(group ID or name*[integer or string]*, delete automatic groups (optional)*[integer]*)
- Get(Name*[string]*)
- GetFromID(ID*[integer]*)
- Total()

Member functions

- Add(Curve*[Curve]*)
- AddAll()
- AddID(ID*[integer]*)
- Contains(Curve*[Curve]*)
- ContainsID(ID*[integer]*)
- GetCurveIDs()
- GetCurves()
- Remove(Curve*[Curve]*)
- RemoveAll()
- RemoveID(ID*[integer]*)
- Spool()
- SpoolID()
- StartSpool()

Group properties

Name	Type	Description
crv_at_ymax	integer	Curve number of the curve with the maximum Y value in the group.
crv_at_ymin	integer	Curve number of the curve with the minimum Y value in the group.
curves (read only)	integer	Number of curves in the group
name (read only)	string	Group name
x_at_ymax	real	X value at the maximum Y value over all curves in the group.
x_at_ymin	real	X value at the minimum Y value over all curves in the group.
x_at_yminpos	real	X value at the minimum positive Y value over all curves in the group.
xmax	real	Maximum X value over all curves in the group.
xmin	real	Minimum X value over all curves in the group.
xminpos	real	Minimum positive X value over all curves in the group.
ymax	real	Maximum Y value over all curves in the group.
ymin	real	Minimum Y value over all curves in the group.
yminpos	real	Minimum positive Y value over all curves in the group.

Detailed Description

The Group class allows you to create, and modify groups. See the documentation below for more details.

Constructor

new Group(name*[string]*)

Description

Create a new Group object.

Arguments

- **name** (string)

Group name used to reference the group

Returns

Group object

Return type

Group

Example

To create a new group with the name X-Velocity

```
var l = new Group("X-velocity");
```

Details of functions

Add(Curve*[Curve]*)

Description

Adds a curve object to group.

Arguments

- **Curve** (Curve)

Curve that will be added to group

Returns

No return value.

Example

To add curve c to curve group g:

```
g.Add(c);
```

AddAll()

Description

Adds all curves to group.

Arguments

No arguments

Returns

No return value.

Example

To add all curves to curve group g:

```
g.AddAll();
```

AddID(ID*[integer]*)

Description

Adds curve by ID to a group.

Arguments

- **ID** (integer)

The ID of the curve you want to add.

Returns

No return value.

Example

To add curve 3 to curve group g:

```
g.AddID(3);
```

Contains(Curve*[Curve]*)

Description

Checks if a curve object is in a curve group.

Arguments

- **Curve** (Curve)

Curve that will be checked

Returns

TRUE if the curve is in the group, otherwise FALSE

Return type

Boolean

Example

To check if a curve object n is in group g

```
var exists = g.Contains(n);
```

ContainsID(ID*[integer]*)

Description

Checks if a curve ID is in a curve group.

Arguments

- **ID** (integer)

The ID of the curve you want to check.

Returns

TRUE if the curve is in the group, otherwise FALSE

Return type

Boolean

Example

To check if a curve ID n is in group g

```
var exists = g.ContainsID(n);
```

DeleteGroup(group ID or name*[integer or string]*, delete automatic groups (optional)*[integer]*) [static]

Description

Deletes a curve group

Arguments

- **group ID or name** (integer or string)

ID of group to delete or name of group. If this argument is 0, delete all groups. Automatically generated groups won't be deleted unless the next argument is set to 1.

- **delete automatic groups (optional)** (integer)

If this argument is 1, automatic groups can be deleted. If no argument or 0, automatic groups cant be deleted.

Returns

No return value

Example

To delete group n

```
Group.DeleteGroup(n);
```

Get(Name*[string]*) [static]

Description

Returns a group object.

Arguments

- **Name** (string)

Name of the group to return object for

Returns

Group object (or Null if the group does not exist).

Return type

Group

Example

To get the group called ’left’

```
var group = Group.Get("left");
```

GetCurveIDs()

Description

Returns an array of Curve ID’s for all the Curves in the group.

Arguments

No arguments

Returns

Array of integers.

Return type

Number

Example

To make an array of Curve ID’s for all the curves in group g:

```
var curves = g.GetCurveIDs();
```

GetCurves()

Description

Returns an array of Curve Objects for all the Curves in the group.

Arguments

No arguments

Returns

Array of Curve objects.

Return type

Array

Example

To make an array of Curve objects for all the curves in group g:

```
var curves = g.GetCurves();
```

GetFromID(ID*[integer]*) [static]

Description

Returns a group object.

Arguments

- **ID** (integer)

ID of the group to return object for

Returns

Group object (or Null if the group does not exist).

Return type

Group

Example

To get the group number 1

```
var group = Group.GetFromID(1);
```

Remove(Curve*[Curve]*)

Description

Removes a curve object from a group.

Arguments

- **Curve** (Curve)

Curve that will be removed from group

Returns

No return value.

Example

To remove curve c from curve group g:

```
g.Remove(c);
```

RemoveAll()

Description

Removes all curves from a group.

Arguments

No arguments

Returns

No return value.

Example

To remove all curves from curve group g:

```
g.RemoveAll();
```

RemoveID(ID*[integer]*)

Description

Remove a curve by ID from a group.

Arguments

- **ID** (integer)

The ID of the curve you want to remove.

Returns

No return value.

Example

To remove curve 3 from curve group g:

```
g.RemoveID(3);
```

Spool()

Description

Spools a group, entry by entry and returns the curve objects. See also Group.StartSpool

Arguments

No arguments

Returns

Curve Object of item, or NULL if no more curves in group

Return type

Curve

Example

To spool group g:

```
var id;
g.StartSpool();
while (id = g.Spool() )
{
    do something...
}
```

SpoolID()

Description

Spools a group, entry by entry and returns the curve ID’s or 0 when no more curves in group. See also Group.StartSpool

Arguments

No arguments

Returns

integer

Return type

Number

Example

To spool group g :

```
var id;
g.StartSpool();
while (id = g.SpoolID() )
{
    do something...
}
```

StartSpool()

Description

Starts a group spooling operation. See also Group.Spool

Arguments

No arguments

Returns

No return value

Example

To start spooling group g:

```
g.StartSpool();
```

Total() [static]

Description

Returns the total number of curve group currently defined

Arguments

No arguments

Returns

Number of curve groups currently defined.

Return type

Number

Example

To get the number of curve groups

```
var total = Group.Total();
```

Include class

The Include class allows you to access the include files in a model. More...

The T/HIS JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Include constants

Constants for Directory separators

Name	Description
Include.NATIVE	Use directory separators native to this machine when writing directory names.
Include.UNIX	Use unix directory separators when writing directory names.
Include.WINDOWS	Use windows directory separators when writing directory names.

Detailed Description

Originally developed for use in PRIMER, the Include class allows a user to create and query include files in a model. A stripped-back version of this class has been added to T/HIS and REPORTER for consistency between the programs.
See File.DriveMapFilename for the current use of this class in T/HIS.

LineStyle class

The LineStyle class contains constants relating to the curve line style. More...

The T/HIS JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

LineStyle constants

Name	Description
LineStyle.DASH	Dashes lines
LineStyle.DASH2	Dash pattern 2
LineStyle.DASH3	Dash pattern 3
LineStyle.DASH4	Dash pattern 4
LineStyle.DASH5	Dash pattern 5
LineStyle.DASH6	Dash pattern 6
LineStyle.NONE	No line
LineStyle.SOLID	Solid lines

Detailed Description

The LineStyle class is used to define the line style used to draw curves:

```
p.style = LineStyle.SOLID;
```

LineWidth class

The LineWidth class contains constants relating to the curve line width. More...

The T/HIS JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

LineWidth constants

Name	Description
LineWidth.BOLD	Bold lines (4 pixels wide)
LineWidth.FINE	Fine lines (1 pixel wide)
LineWidth.HEAVY	Heavy lines (8 pixels wide)
LineWidth.NORMAL	Normal lines (2 pixels wide)
LineWidth.W1	1 pixel wide
LineWidth.W10	10 pixel wide
LineWidth.W2	2 pixel wide
LineWidth.W3	3 pixel wide
LineWidth.W4	4 pixel wide
LineWidth.W5	5 pixel wide
LineWidth.W6	6 pixel wide
LineWidth.W7	7 pixel wide
LineWidth.W8	8 pixel wide
LineWidth.W9	9 pixel wide

Detailed Description

The LineWidth class is used to define the line width used to draw curves:

```
p.width = LineWidth.NORMAL;
```

Link class

The Link class allows you to link to other Oasys LS-DYNA Environment programs on the computer. More...

The T/HIS JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- ConnectPending()

Member functions

- Send(data*[ArrayBuffer or string]*)
- Terminate()

Link constants

Constants for programs

Name	Description
Link.D3PLOT	Link to D3PLOT
Link.PRIMER	Link to PRIMER
Link.THIS	Link to T-HIS

Constants for status

Name	Description
Link.CONNECTED	Link is connected to the remote script
Link.CREATED	Link has been created but not connected to the remote script
Link.DISCONNECTED	Link has been disconnected as the remote script has finished/exited
Link.TERMINATED	Link has been terminated as the remote program has closed/crashed

Link properties

Name	Type	Description
onDisconnect	function	Function to call when when the link is disconnected, either because the remote program has terminated, or the remote script has finished. The status property can be used to distinguish these cases. The function has no arguments. The Link object is accessible in the function using the 'this' keyword. To unset the function, set the property to null.

onReceipt	function	Function to call to acknowledge receipt of the data in the remote program (data sent from a Send command). The function takes a single argument which is the UUID returned from the original Send command. Note that this function will only be called if the link is asynchronous. In the synchronous case the local program will wait for the remote program to process the Send command so this is unnecessary. To unset the function, set the property to null.
onReceive	function	Function to call when data is received from a Send command in the remote program. The function takes a single argument which contains the data passed by Send. To unset the function, set the property to null.
size (read only)	integer	Size of shared memory buffer in KB
status (read only)	constant	The status of the connection. Can be Link.CREATED, Link.CONNECTED, Link.DISCONNECTED or Link.TERMINATED
synchronous	boolean	whether link Send commands are synchronous or asynchronous (default)
timeout	integer	Timeout for Send commands in msecs. If equal to 0 (default) then a default timeout of 5s is used

Detailed Description

The Link class gives you simple functions for a script to pass information to and from scripts running on the other Oasys LS-DYNA Environment programs on the same computer.
It is the JavaScript equivalent of the user pressing the 'Post' button in PRIMER to start D3PLOT or T/HIS, or the user pressing 'PRIMER' from D3PLOT or T/HIS to start PRIMER, and then passing data between the programs. If the user has not already started the remote program by doing this manually, the Link class will start the remote program.
The following simple example shows a script that is running in PRIMER, starting a link to D3PLOT. Once the link is established, the scripts running in PRIMER and D3PLOT can pass string data to the other program. This data is then displayed in a window on the remote program.
Note that just like when starting the link from PRIMER to D3PLOT manually, to start the link from the script, a model must already be loaded into PRIMER that also has results files for D3PLOT.
Script 'primer.js' running in PRIMER:

```
// Global scope for link variable
var link;
// Get path for this script, and hence path for d3plot.js
var scriptD3PLOT = arguments[0].replace('primer.js', 'd3plot.js');
// Create a simple GUI
Window.Theme(Window.THEME_CURRENT);
var w = new Window("Link PRIMER", 0.9, 1.0, 0.9, 1.0);
var start = new Widget(w, Widget.BUTTON, 1, 40, 1, 7, "Start D3PLOT");
start.onClick = startD3PLOT;
var msg  = new Widget(w, Widget.TEXTBOX,  1, 50, 8, 14, "");
var send = new Widget(w, Widget.BUTTON,  51, 90, 8, 14, "Send Message");
send.onClick = sendMessage;
send.active = false;
w.onClose = Exit;
w.Show(false);
////////////////////////////////////////////////////////////////////////////
function startD3PLOT()
{
// Function to start D3PLOT and create link
// object to pass to Link constructor
    var obj = {
               script:             scriptD3PLOT,
               connectFunction:    "connectD3PLOT",
               receiveFunction:    "receiveD3PLOT",
               disconnectFunction: "Exit",
               onReceive:          receivePRIMER,
               onDisconnect:       Exit,
              };
// Create link and save to global scope for this simple example
    link = new Link(Link.D3PLOT, obj);
// Do NOT try to immediately send something to D3PLOT using the link
// It may not be properly initialised yet.
// See functions connectD3PLOT and receivePRIMER
// Make the 'Send Message' button active and disable the 'Start D3PLOT' button
```

```
    start.active = false;
    send.active  = true;
}
////////////////////////////////////////////////////////////////////////////
function receivePRIMER(data)
{
// Function called when data is received from D3PLOT across the link
// If the object has a property called 'connect' then D3PLOT has sent
// this from the connect function and so is ready to receive data.
    if (data.connect)
    {
        Message("D3PLOT is connected");
// We could now send information to D3PLOT if needed
//      link.Send(...);
    }
// If the object has a property called 'string' then it is the string
// from the message box
    else if (data.string)
    {
        Window.Message("Example string from D3PLOT", data.string );
    }
}
////////////////////////////////////////////////////////////////////////////
function sendMessage()
{
// Sends whatever is in textbox when 'Send Message' is pressed, to D3PLOT
    var uuid = link.Send( { string: msg.text } );
}
```

Script 'd3plot.js' running in D3PLOT:

```
// Global scope for link variable
var link;
// Create a simple GUI
Window.Theme(Window.THEME_CURRENT);
var w = new Window("Link D3PLOT", 0.9, 1.0, 0.9, 1.0);
var msg  = new Widget(w, Widget.TEXTBOX,  1, 50, 1, 7, "");
var send = new Widget(w, Widget.BUTTON,  51, 90, 1, 7, "Send Message");
send.onClick = sendMessage;
send.active  = false;
w.onClose = Exit;
w.Show(false);
////////////////////////////////////////////////////////////////////////////
function connectD3PLOT(l)
{
// Function called when the connection from PRIMER is made
// Save link to global scope for this simple example
    link = l;
// Make the 'Send Message' button active
    send.active = true;
// Send a command back to PRIMER to say that we are connected
    link.Send( { connect: "CONNECTED" } );
}
////////////////////////////////////////////////////////////////////////////
function receiveD3PLOT(data)
{
// Function called when data is received from PRIMER across the link
// If the object has a property called 'string' then it is the string
// from the message box
    if (data.string)
    {
        Window.Message("Example string from PRIMER", data.string);
    }
}
////////////////////////////////////////////////////////////////////////////
function sendMessage()
{
// Sends whatever is in textbox when 'Send Message' is pressed, to PRIMER
    var uuid = link.Send( { string: msg.text } );
}
```

Constructor

new Link(program*[constant]*, data*[object]*)

Description

Create a new Link object to communicate with one of the linked Oasys LS-DYNA programs running on the computer, starting it if required.
Note that you should **not** try to use the Send method immediately after the constructor. If you do, the remote program may not have time to initialise properly. If you want to interact with the remote program immediately then the connect function in the remote program should send a command to the local program to say that it is initialised. The onReceive function in the local program can then be used to send the command. See the simple example at the top of this class for more details.
For the specific case of D3PLOT starting T/HIS, the connect function will not be called straight away. As D3PLOT and T/HIS are integrated together into a single user interface, the script **must** return back to the main event loop after the constructor is called so that D3PLOT and T/HIS can communicate with each other to set up the integration. Once this has been done and D3PLOT is 'idle', the connect function will be called. The idleTime property in the data object can be used to control this delay if required. As D3PLOT must return back to the event loop when starting T/HIS do not put any statements immediately after the link constructor call.

Arguments

- **program** (constant)

The program that you want to link to. See the programs constants

- **data** (object)

link data

Object has the following properties:

Name	Type	Description
connectFunction	string	Name of function to call in the remote script when the link connected. This function will be called when the connection is made with a Link object as the argument.
disconnectFunction (optional)	string	Name of function to call in the remote script when the link is disconnected, either because the local program has terminated, or the local script has finished. The status property can be used to distinguish these cases. This is provided as an easy way to set up a disconnect callback in the remote script. Alternatively, the remote script could set the onDisconnect property in connectFunction.
idelTime (optional)	integer	For the specific case of starting T/HIS from D3PLOT, the required time for D3PLOT to be idle for in msecs before the connect function is called. If omitted then a default idle time of 1s is used
model (optional)	integer	Model number to pass across the link to the remote program
onDisconnect (optional)	function	Function to call in the local script when the link is disconnected, either because the remote program has terminated, or the remote script has finished. The status property can be used to distinguish these cases.
onReceipt (optional)	function	Function to call in the local script to acknowledge receipt of the data in the remote program (data sent from a Send command in the local program to the remote program). The function takes a single argument which is the UUID returned from the original Send command. This function is useful if the link is asynchronous to know that the command has been processed. In the synchronous case the local program will wait for the remote program to process the Send command so this is unnecessary. However, the function can still be used.
onReceive (optional)	function	Function to call in the local script when data is received from a Send command in the remote script. The function takes a single argument which contains the data passed by Send
path (optional)	string	Path to a model to pass across the link to the remote program. This can be used instead of a model. For example if you want to open a different model in D3PLOT to what you have open in PRIMER then you could pass the path to the d3plot file to open

receiptFunction (optional)	string	Name of function to call in the remote script to acknowledge receipt of the data in the local script (data sent from a Send command in the remote link object to the local script). The function takes a single argument which is the UUID returned from the original Send command. Note that this function will only be called if the link is asynchronous. In the synchronous case the local program will wait for the remote to process the Send command so this is unnecessary. This is provided as an easy way to set up a receipt callback in the remote script. Alternatively, the remote script could set the onReceipt property in connectFunction.
receiveFunction (optional)	string	Name of function to call in the remote script when data is received from a Send command in the local script. The function takes a single argument which contains the data passed by Send. This is provided as an easy way to set up a receive callback in the remote script. Alternatively, the remote script could set the onReceive property in connectFunction.
script	string	The script to run
size (optional)	integer	Size of shared memory buffer in KB. If omitted a default size of 10KB is used
synchronous (optional)	boolean	Whether link Send commands are synchronous or asynchronous (default if omitted)
timeout (optional)	integer	Timeout for Send commands in msecs. If equal to 0 (default if omitted) then a default timeout of 5s is used

Returns

Link object

Return type

Link

Example

To create a connection to D3PLOT, calling function 'myConnectFunc' in remote script '/example/script.js' when the connection is made:

```
var s = new Link(Link.D3PLOT, { script: '/example/script.js', connectFunction:
'myConnectFunc' } );
```

Details of functions

ConnectPending() [static]

Description

Check if the script has been started on the remote side of a link and, if so, whether calling the connect function is pending. This may be useful to make a script behave differently, depending on whether it is being run normally or has been started on the remote side of a link.

Arguments

No arguments

Returns

true if a remote link script is waiting to connect, false for all other cases

Return type

boolean

Example

To check if the script has been started on the remote side of a link:

```
if (Link.ConnectPending() ) { do something ... }
```

Send(data*[ArrayBuffer or string]*)

Description

Send data/command across link

Arguments

- **data** (ArrayBuffer or string)

The data/command to pass. The data can either be an ArrayBuffer or any JavaScript type that can be represented as JSON

Returns

UUID string to uniquely identify this Send command

Return type

String

Example

To send some data across link l:

```
var data = { action: "read", filename: "example.ptf" };
l.Send(data);
```

Terminate()

Description

Terminate the program on the remote side of the link

Arguments

No arguments

Returns

No return value

Example

To terminate the program on the remote side of link l:

```
l.Terminate();
```

Model class

The Model class gives you access to models in T/HIS. More...

The T/HIS JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Exists(model number*[integer]*)
- GetFromID(model number*[integer]*)
- HighestID()
- Read(filename*[string]*, filetype (optional)*[integer]*)
- Total()

Member functions

- ClearFlag(flag*[Flag]*, entity_type*[integer]*, item*[integer]*, end (optional)*[integer]*)
- Delete()
- FlagAll(flag*[Flag]*, entity_type*[integer]*)
- Flagged(flag*[Flag]*, entity_type*[integer]*, item*[integer]*)
- GetDataFlagged(flag*[Flag]*, data_comp*[integer]*, int_pnt (optional)*[object | integer]*, extra (optional)*[integer]*)
- GetInternalID(entity_type*[integer]*, item*[integer]*)
- GetLabel(entity_type*[integer]*, item*[integer]*)
- GetLabelFromName(entity_type*[integer]*, name*[string]*)
- GetModelUnits()
- GetName(entity_type*[integer]*, item*[integer]*)
- GetNumberFlagged(flag*[Flag]*, entity_type (optional)*[integer]*)
- GetNumberOf(entity_type*[integer]*)
- QueryDataPresent(data_comp*[integer]*, entity_type (optional)*[integer]*, int_pnt (optional)*[object | integer]*, extra (optional)*[integer]*)
- SetFlag(flag*[Flag]*, entity_type*[integer]*, item*[integer]*, end (optional)*[integer]*)
- SetModelUnits(unit system*[String]*)
- UnflagAll(flag*[Flag]*, entity_type*[integer]*)

Model constants

Name	Description
Model.ALL_FILES	Option to select all files (.thf, LSDA, ASCII, .ztf) when reading model in.
Model.ASCII	Option to select ASCII files when reading model in.
Model.LSDA	Option to select LSDA/binout file when reading model in.
Model.THF	Option to select .thf/d3thdt file when reading model in.
Model.XTF	Option to select .xtf/xtfile file when reading model in.
Model.ZTF	Option to select .ztf file when reading model in.

Model properties

Name	Type	Description
dir (read only)	string	Directory containing the model file.
file (read only)	string	File selected when reading the model.

id (read only)	integer	Model ID
title (read only)	string	Model title.

Detailed Description

The Model class contains information on filenames and directories belonging to a model. See the documentation below for more details.

Details of functions

ClearFlag(flag*[Flag]*, entity_type*[integer]*, item*[integer]*, end (optional)*[integer]*)

Description

Clears a defined flag on an internal (or external) item(s) of type of entity_type in the model.

Arguments

- **flag** (Flag)

The flag you want to clear.

- **entity_type** (integer)

The Entity type that the defined flag will be cleared on.

- **item** (integer)

If +ive: The internal item number starting from 1. If -ive: The external item label.

- **end (optional)** (integer)

To unflag range of items, specify an optional end of range. Unflags items from item to range.

Returns

TRUE if the flag is successfully cleared on the item, otherwise FALSE

Return type

Boolean

Example

To clear the flag f on the 6th node in model m:

```
m.ClearFlag(f, Entity.NODE, 6);
```

To clear the flag f on the Node 13456 in model m:

```
m.ClearFlag(f, Entity.NODE, -13456);
```

To clear the flag f on the first 10 nodes in model m:

```
m.ClearFlag(f, Entity.NODE, 1, 10);
```

To clear the flag f on nodes with labels 1000, 1001, 1002, ..., 1009 in model m:

```
m.ClearFlag(f, Entity.NODE, -1000, -1009);
```

Delete()

Description

Deletes a model
Do not use the Model object after calling this method.

Arguments

No arguments

Returns

TRUE if the model sucessfully deleted, otherwise FALSE

Return type

Boolean

Example

To delete model m:

```
var deleted = m.Delete();
```

Exists(model number*[integer]*) [static]

Description

Checks if a model exists

Arguments

- **model number** (integer)

The number of the model you want to check the existence of.

Returns

TRUE if the model exists, otherwise FALSE

Return type

Boolean

Example

To check if a model n exists

```
var exists = Model.Exists(n);
```

FlagAll(flag*[Flag]*, entity_type*[integer]*)

Description

Sets a defined flag on all of items of type of entity_type in the model.

Arguments

- **flag** (Flag)

The flag you want to set.

- **entity_type** (integer)

The Entity type that the defined flag will be set on.

Returns

TRUE if the flag is successfully set on all the items, otherwise FALSE

Return type

Boolean

Example

To set the flag f on all the nodes in model m:

```
m.FlagAll(f, Entity.NODE);
```

Flagged(flag*[Flag]*, entity_type*[integer]*, item*[integer]*)

Description

Checks if a defined flag is set on an internal (or external) item of type of entity_type in the model.

Arguments

- **flag** (Flag)

The flag you want to check.

- **entity_type** (integer)

The Entity type to check.

- **item** (integer)

If +ive: The internal item number starting from 1. If -ive: The external item label.

Returns

TRUE if the flag is set, FALSE if the flag is not set.

Return type

Boolean

Example

To check if flag f is set on the 6th node in model m:

```
m.Flagged(f, Entity.NODE, 6);
```

To check if flag f is set on the Node 13456 in model m:

```
m.Flagged(f, Entity.NODE, -13456);
```

GetDataFlagged(flag*[Flag]*, data_comp*[integer]*, int_pnt (optional)*[object | integer]*, extra (optional)*[integer]*)

Description

Gets curve objects for a data component for relevant items that are flagged with a specified flag in the model.
Some data components are valid for different entity types (e.g. SXX). If the same flag is set on items of different entity types, data is returned for all relevant, flagged entity types.
To return the same data for multiple items of the same type, it will be much faster if you flag all items you want data for, and do a single call to GetDataFlagged().
The curves are ordered by type, then by the ascending internal index of the items. Use curve properties to identify which curve is which. **If the data is not available in the model for a flagged item, or not available for the selected integration points or extra value, a curve is not returned.** You can use QueryDataPresent() to check if the data is available.
It is recommended that you check the number of curves returned. This can be compared with the number of flagged entities, see GetNumberFlagged().
If the data is generally available in the model, but not for the specific flagged item, a "null curve" which contains no x-y data values is returned. For example, a specific shell may have fewer integration points than MAX_INT for all shells, a "null curve" would be returned for the higher integration points.

Arguments

- **flag** (Flag)

The flag to use. For model data, use 0 to define a null "padding" argument.

- **data_comp** (integer)

The Data Component to extract.

- **int_pnt (optional)** (object) | integer

The integration points to extract.
This argument can be either an integer or an object.
This argument is ignored when the entity type is not SOLID, SHELL, THICK_SHELL or BEAM.

An *integer* specifies the integration point to extract:
For SOLIDs: value between 0 for Average/Centre and 8. (Defaults to Average/Centre).
For SHELLs and THICK_SHELLs: value between 1 and # integration points, or codes Constant.TOP, Constant.MIDDLE, Constant.BOTTOM. (Defaults to MIDDLE integration point).
For integrated BEAMs: value between 1 and # integration points. (Defaults to integration point 1).

Use 0 to define a null "padding" argument, then uses the default integration point.

Object has the following properties:

Name	Type	Description
ip	integer	Through thickness integration point as described above.
np (optional)	integer	The nodes to extrapolate to. For SOLIDs, SHELLs and THICK_SHELLs: value between 1 and # nodes on the entity. (Defaults to none).
op (optional)	integer	On plan integration point. For SHELLs and THICK_SHELLs: value between 0 for Average/Centre and 4. (Defaults to Average/Centre).

- **extra (optional)** (integer)

The extra component id for SOLIDs, SHELLs, THICK_SHELLs or BEAMs.

Returns

Array of Curve objects.

Return type

Array

Example

To get X direct stress for flagged SOLIDs, SHELLs and THICK_SHELLs with flag f in model m:

```
var cur_array = m.GetDataFlagged(f, Component.SXX);
```

To get X direct stress at top integration point for flagged SHELLs and THICK_SHELLs with flag f in model m:

```
var cur_array = m.GetDataFlagged(f, Component.SXX, Constant.TOP);
```

To get X direct stress at top integration point, and on-plan integration point 3 for flagged SHELLs and THICK_SHELLs with flag f in model m:

```
var cur_array = m.GetDataFlagged(f, Component.SXX, {ip:Constant.TOP, op:3});
```

To get extra beam data 3 for flagged BEAMs with flag f in model m:

```
var cur_array = m.GetDataFlagged(f, Component.BEX, 0, 3);
```

To get the total mass in model m:

```
var cur_array = m.GetDataFlagged(0, Component.GMASS);
```

GetFromID(model number*[integer]*) [static]

Description

Returns the Model object for a model ID or null if model does not exist.

Arguments

- **model number** (integer)

number of the model you want the Model object for

Returns

Model object (or null if model does not exist).

Return type

Model

Example

To get the model n

```
var model = Model.GetFromID(n);
```

GetInternalID(entity_type*[integer]*, item*[integer]*)

Description

Gets the internal ID of external item of type entity_type in the model.

Arguments

- **entity_type** (integer)

The Entity type of the item.

- **item** (integer)

The external item number.

Returns

Integer internal ID (starting from 1) with reference to the entity_type code. Returns integer internal ID of 0 if item cannot be found.

Return type

Number

Example

To get the internal ID of Airbag 300 in model m:

```
var x = m.GetInternalID(Entity.AIRBAG, 300);
```

GetLabel(entity_type*[integer]*, item*[integer]*)

Description

Gets the external label of internal item of type entity_type in the model.

Arguments

- **entity_type** (integer)

The Entity type of the item.

- **item** (integer)

The internal item number starting from 1.

Returns

Integer external ID (or 0 if there is an error, or the internal ID if there are no external IDs).

Return type

Number

Example

To get the external ID of the 2nd airbag in model m:

```
var x = m.GetLabel(Entity.AIRBAG, 2);
```

GetLabelFromName(entity_type*[integer]*, name*[string]*)

Description

Gets the external label from the database history name name of type entity_type in the model. This is quicker if you use parent entity type codes (e.g. Entity.WELD rather than Entity.WELD_CONSTRAINED)

Arguments

- **entity_type** (integer)

The Entity type of the item.

- **name** (string)

The name of the item. If only the first part of the name is given, it must be unambiguous.

Returns

Integer external ID of the first matching name (or 0 if there is an error).

Return type

Number

Example

To get the external label the of Contact named "Rear Bolt" in database history:

```
var name = m.GetLabelFromName(Entity.CONTACT, "Rear Bolt");
```

GetModelUnits()

Description

Returns the Model units of a particular model

Arguments

No arguments

Returns

String indicating the model unit system of the model

Return type

String

Example

To get the model units of model m

```
var units = m.GetModelUnits();
```

GetName(entity_type*[integer]*, item*[integer]*)

Description

Gets the database history name of an internal (or external) item of type entity_type in the model.

Arguments

- **entity_type** (integer)

The Entity type of the item.

- **item** (integer)

If +ive: The internal item number starting from 1. If -ive: The external item label.

Returns

String containing the database history name (or null if not available).

Return type

String

Example

To get the database history name of the 2nd airbag in model m:

```
var name = m.GetName(Entity.AIRBAG, 2);
```

To get the database history name of Airbag 300 in model m:

```
var name = m.GetName(Entity.AIRBAG, -300);
```

GetNumberFlagged(flag*[Flag]*, entity_type (optional)*[integer]*)

Description

Gets the number of entities flagged with a requested flag in the model.

Arguments

- **flag** (Flag)

The flag you want to check.

- **entity_type (optional)** (integer)

If specified, the Entity type to look at. If not specified, all types are looked at.

Returns

Integer number

Return type

Number

Example

To get the number of airbag parts flagged with flag f in model m:

```
var num = m.GetNumberFlagged(f, Entity.AIRBAG_PART_DATA);
```

GetNumberOf(entity_type*[integer]*)

Description

Gets the number of entities of a requested type in the model.

Arguments

- **entity_type** (integer)

The Entity type that you want to know the number of.

Returns

Integer number

Return type

Number

Example

To get the number of airbags in model m:

```
var num = m.GetNumberOf(Entity.AIRBAG);
```

HighestID() [static]

Description

Returns the ID of the highest model currently being used

Arguments

No arguments

Returns

ID of highest model currently being used.

Return type

Number

Example

To get the highest model ID

```
var id= Model.HighestID();
```

QueryDataPresent(data_comp*[integer]*, entity_type (optional)*[integer]*, int_pnt (optional)*[object | integer]*, extra (optional)*[integer]*)

Description

Checks if a data component data_comp for a given entity is present in a model's database. For SOLIDs, SHELLs, THICK_SHELLs and BEAMs the integration point and extra component ID can also be checked. This will show if curves for any flagged items of this type will be returned for GetDataFlagged(). Note, it does not check if the data component is *valid*, for example a specific shell may have fewer integration points than MAX_INT for all shells, so curves returned for GetDataFlagged() may still be "null" with no x-y data.

Arguments

- **data_comp** (integer)

The Data Component to check.

- **entity_type (optional)** (integer)

The Entity type to check. This argument can only be omitted when checking for global model data.

- **int_pnt (optional)** (object) | integer

The integration points to check.
This argument can be either an integer or an object.
This argument is ignored if the entity type is not SOLID, SHELL, THICK_SHELL or BEAM.

An *integer* specifies the integration point to check:
For SOLIDs: value between 0 for Average/Centre and 8. (Defaults to Average/Centre).
For SHELLs and THICK_SHELLs: value between 1 and # integration points, or codes Constant.TOP, Constant.MIDDLE, Constant.BOTTOM. (Defaults to MIDDLE integration point).
For integrated BEAMs: value between 1 and # integration points. (Defaults to integration point 1).

Use 0 to define a null "padding" argument, then checks the default integration point.

Object has the following properties:

Name	Type	Description
ip	integer	Through thickness integration point as described above.
np (optional)	integer	The nodes to extrapolate to. For SOLIDs, SHELLs and THICK_SHELLs: value between 1 and # nodes on the entity. (Defaults to none).
op (optional)	integer	On plan integration point. For SHELLs and THICK_SHELLs: value between 0 for Average/Centre and 4. (Defaults to Average/Centre).

- **extra (optional)** (integer)

The extra component id for SOLIDs, SHELLs, THICK_SHELLs or BEAMs.

Returns

true if data is present, otherwise false.

Return type

Boolean

Example

To check for X direct stress data for SOLIDs in model m:

```
if(m.QueryDataPresent(Component.SXX, Entity.SOLID)) ...
```

To check for X direct stress data at integration point 5 for SHELLs in model m:

```
if(m.QueryDataPresent(Component.SXX, Entity.SHELL, 5)) ...
```

To check for X direct stress data at both the top integration point, and also extrapolated to node 3 for SHELLs in model m:

```
if(m.QueryDataPresent(Component.SXX, Entity.SHELL, {ip:Constant.TOP, np:3}) ...
```

To check for extra 3 beam data for BEAMs in model m:

```
if(m.QueryDataPresent(Component.BEX, Entity.BEAM, 0, 3);
```

To check for total mass data in model m:

```
if(m.QueryDataPresent(Component.GMASS));
```

Read(filename*[string]*, filetype (optional)*[integer]*) [static]

Description

Reads in a new model.

Arguments

- **filename** (string)

Filename you want to read.

- **filetype (optional)** (integer)

Filetypes you want to read. Can be bitwise OR of Model.THF, Model.XTF, Model.LSDA, Model.ASCII, Model.ZTF and Model.ALL_FILES. If omitted all available files will be read.

Returns

Model object (or null if error).

Return type

Model

Example

To read in model /data/test/file.thf:

```
var m = Model.Read("/data/test/file.thf");
```

To read in model /data/test/file.thf only with .ztf files:

```
var m = Model.Read("/data/test/file.thf", Model.ZTF);
```

To read in model /data/test/file.thf only with .thf, .ascii and .ztf files:

```
var m = Model.Read("/data/test/file.thf", Model.THF | Model.ASCII | Model.ZTF);
```

SetFlag(flag*[Flag]*, entity_type*[integer]*, item*[integer]*, end (optional)*[integer]*)

Description

Sets a defined flag on an internal (or external) item(s) of type of entity_type in the model.

Arguments

- **flag** (Flag)

The flag you want to set.

- **entity_type** (integer)

The Entity type that the defined flag will be set on.

- **item** (integer)

If +ive: The internal item number starting from 1. If -ive: The external item label.

- **end (optional)** (integer)

To flag range of items, specify an optional end of range. Flags items from item to range.

Returns

TRUE if the flag is successfully set on the item, otherwise FALSE

Return type

Boolean

Example

To set the flag f on the 6th node in model m:

```
m.SetFlag(f, Entity.NODE, 6);
```

To set the flag f on the Node 13456 in model m:

```
m.SetFlag(f, Entity.NODE, -13456);
```

To set the flag f on the first 10 nodes in model m:

```
m.SetFlag(f, Entity.NODE, 1, 10);
```

To set the flag f on nodes with labels 1000, 1001, 1002, ..., 1009 in model m:

```
m.SetFlag(f, Entity.NODE, -1000, -1009);
```

SetModelUnits(unit system*[String]*)

Description

Set the model units of a model to the units provided by the user

Arguments

- **unit system** (String)

The unit system you want to set the model units of model to

Returns

TRUE if the Model units are set successfully else FALSE

Return type

Boolean

Example

To set the model units of model m to "U2"

```
m.SetModelUnits("U2");
```

Total() [static]

Description

Returns the total number of models.

Arguments

No arguments

Returns

integer

Return type

Number

Example

To find how many models there are in T/HIS:

```
var num = Model.Total();
```

UnflagAll(flag*[Flag]*, entity_type*[integer]*)

Description

Unsets a defined flag flag on all of items of type of entity_type in the model.

Arguments

- **flag** (Flag)

The flag you want to unset.

- **entity_type** (integer)

The Entity type that the defined flag will be unset on.

Returns

TRUE if the flag is successfully unset on all the items, otherwise FALSE

Return type

Boolean

Example

To unset the flag f on all the nodes in model m:

```
m.UnflagAll(f, Entity.NODE);
```

Operate class

The Operate class gives you access to the built in curve operations in T/HIS. More...

The T/HIS JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Abs(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*)
- Acos(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*)
- Acu(Input Curve*[Curve]*, Offset*[real]*, Time Period*[real]*, Output Curve (optional)*[Curve]*)
- Ad(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*)
- Add(Input Curve*[Curve]*, Second Curve or constant*[Curve or real]*, Output Curve (optional)*[Curve]*)
- Adx(First Curve*[Curve]*, Second Curve or constant*[Curve or real]*, Output Curve (optional)*[Curve]*)
- Asi(X Acceleration*[Curve]*, Y Acceleration*[Curve]*, Z Acceleration*[Curve]*, Acceleration conversion factor*[real]*, X Acceleration Limit*[real]*, Y Acceleration Limit*[real]*, Z Acceleration Limit*[real]*, Calculation method*[string]*, X axis interval (optional)*[real]*, Output Curve (optional)*[Curve]*)
- Asin(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*)
- Atan(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*)
- Atan2(First Input Curve*[Curve]*, Second Input Curve*[Curve]*, Output Curve (optional)*[Curve]*)
- Av(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*)
- Ave(Curves*[Array of Curve objects]*, Output Curve (optional)*[Curve]*)
- Bes(Input Curve*[Curve]*, Frequency*[real]*, Order*[integer]*, X axis interval (optional)*[real]*, Output Curve (optional)*[Curve]*)
- Blc(Input Curve*[Curve]*)
- But(Input Curve*[Curve]*, Frequency*[real]*, Order*[integer]*, X axis interval (optional)*[real]*, Output Curve (optional)*[Curve]*)
- C1000(Input Curve*[Curve]*, X axis interval (optional)*[real]*, Output Curve (optional)*[Curve]*)
- C180(Input Curve*[Curve]*, X axis interval (optional)*[real]*, Output Curve (optional)*[Curve]*)
- C60(Input Curve*[Curve]*, X axis interval (optional)*[real]*, Output Curve (optional)*[Curve]*)
- C600(Input Curve*[Curve]*, X axis interval (optional)*[real]*, Output Curve (optional)*[Curve]*)
- Cat(First Curve*[Curve]*, Second Curve*[Curve or real]*, Output Curve (optional)*[Curve]*)
- Clip(Input Curve*[Curve]*, X min*[real]*, X max*[real]*, Y min*[real]*, Y max*[real]*, Output Curve (optional)*[Curve]*)
- Com(First Curve*[Curve]*, Second Curve*[Curve or real]*, Output Curve (optional)*[Curve]*)
- Cor(First Curve*[Curve]*, Second Curve*[Curve]*, Correlation type*[string]*)
- Cor3(First Curve*[Curve]*, Second Curve*[Curve]*, X axis factor (optional)*[real]*, Y axis factor (optional)*[real]*)
- Cos(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*)
- Da(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*)
- Dif(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*)
- Div(First Curve*[Curve]*, Second Curve or constant*[Curve or real]*, Output Curve (optional)*[Curve]*)
- Dix(First Curve*[Curve]*, Second Curve or constant*[Curve or real]*, Output Curve (optional)*[Curve]*)
- Dmg(Head Rotation Velocity X*[Curve]*, Head Rotation Velocity Y*[Curve]*, Head Rotation Velocity Z*[Curve]*, Calculation method*[string]*, X axis interval (optional)*[real]*, Filter Class (optional)*[string]*)
- Ds(Input Curve*[Curve]*, Broadening Factor*[real]*, Redefine Frequencies*[string]*, Output Curve (optional)*[Curve]*)
- Dv(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*)
- Env(Curves*[Array of Curve objects]*, Output Curve (optional)*[Curve]*)
- Err(First Curve*[Curve]*, Second Curve*[Curve or real]*, Output Curve (optional)*[Curve]*)
- Exc(Input Curve*[Curve]*, Output option*[string]*, Output Curve (optional)*[Curve]*)
- Exp(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*)
- Fft(Input Curve*[Curve]*, Output option*[string]*, X axis interval (optional)*[real]*, Scaling option (optional)*[string]*)
- Fir(Input Curve*[Curve]*, X axis interval (optional)*[real]*, Output Curve (optional)*[Curve]*)
- Hic(Input Curve*[Curve]*, Window*[real]*, Acceleration factor*[real]*)
- Hicd(Input Curve*[Curve]*, Window*[real]*, Acceleration factor*[real]*)
- Ifft(First Curve*[Curve]*, Second Curve*[Curve]*, Input type*[string]*)
- Int(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*)
- Log(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*)

- Log10(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*)
- Log10x(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*)
- Logx(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*)
- Lsq(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*)
- Map(First Curve*[Curve]*, Second Curve*[Curve or real]*, Output Curve (optional)*[Curve]*)
- Max(Curves*[Array of Curve objects]*, Output Curve (optional)*[Curve]*)
- Min(Curves*[Array of Curve objects]*, Output Curve (optional)*[Curve]*)
- Mon(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*)
- Mul(First Curve*[Curve]*, Second Curve or constant*[Curve or real]*, Output Curve (optional)*[Curve]*)
- Mux(First Curve*[Curve]*, Second Curve or constant*[Curve or real]*, Output Curve (optional)*[Curve]*)
- Ncp(First Curve*[Curve]*, Second Curve*[Curve]*)
- Nij(Shear Force*[Curve]*, Axial Force*[Curve]*, Moment*[Curve]*, Fzc_t*[real]*, Fzc_c*[real]*, Myc_f*[real]*, Myc_e*[real]*, E*[real]*)
- Nor(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*)
- Nor2(Input Curve*[Curve]*, Y Min Value*[real]*, Y Max Value*[real]*, Lock to Axis Y Min*[integer]*, Lock to Axis Y Max*[integer]*, Output Curve (optional)*[Curve]*)
- Nox(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*)
- Nox2(Input Curve*[Curve]*, X Min Value*[real]*, X Max Value*[real]*, Lock to Axis X Min*[integer]*, Lock to Axis X Max*[integer]*, Output Curve (optional)*[Curve]*)
- Octave(Input Curve*[Curve]*, Band type to convert to*[String]*, Output Type*[String]*, Input Type*[String]*, Output Curve (optional)*[Curve]*)
- Olc(Input Curve*[Curve]*, Second Curve or constant*[Curve or real]*, X axis interval (optional)*[real]*, Filter Class (optional)*[string]*)
- Order(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*)
- Pbut(Input Curve*[Curve]*, Frequency*[real]*, Order*[integer]*, X axis interval (optional)*[real]*, Output Curve (optional)*[Curve]*)
- Power(Input Curve*[Curve]*, Power*[real]*, Output Curve (optional)*[Curve]*)
- Rave(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*)
- Rec(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*)
- Reg(Input Curve*[Curve]*, X axis interval*[real]*, Output Curve (optional)*[Curve]*)
- Res(Curves*[Array of Curve objects]*, Output Curve (optional)*[Curve]*)
- Rev(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*)
- Rs(Input Curve*[Curve]*, Damping Factor*[real]*, Sampling Points*[int]*, X axis interval (optional)*[real]*, Output Curve (optional)*[Curve]*)
- Sin(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*)
- Smooth(Input Curve*[Curve]*, Smoothing Factor*[integer]*, Output Curve (optional)*[Curve]*)
- Sqr(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*)
- Stress(Input Curve*[Curve]*, Convert to*[string]*, Output Curve (optional)*[Curve]*)
- Sub(First Curve*[Curve]*, Second Curve or constant*[Curve or real]*, Output Curve (optional)*[Curve]*)
- Sum(Curves*[Array of Curve objects]*, Output Curve (optional)*[Curve]*)
- Sux(First Curve*[Curve]*, Second Curve or constant*[Curve or real]*, Output Curve (optional)*[Curve]*)
- Tan(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*)
- Thiv(X Acceleration*[Curve]*, Y Acceleration*[Curve]*, Yaw Rate*[Curve]*, Dx*[real]*, Dy*[real]*, X0*[real]*)
- Ti(Axial Force*[Curve]*, X Moment*[Curve]*, Y Moment*[Curve]*, Fzc*[real]*, Mrc*[real]*, X axis interval (optional)*[real]*, Filter Class (optional)*[string]*, Output Curve (optional)*[Curve]*)
- Tms(Input Curve*[Curve]*, Period*[real]*)
- Translate(Input Curve*[Curve]*, X value*[real]*, Y value*[real]*, Output Curve (optional)*[Curve]*)
- Tti(Upper Rib Acceleration*[Curve]*, Lower Rib Acceleration*[Curve]*, T12 Acceleration*[Curve]*)
- Va(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*)
- Vc(Input Curve*[Curve]*, A*[real]*, B*[real]*, Calculation method*[string]*, Output Curve (optional)*[Curve]*)
- Vd(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*)
- Vec(First Curve*[Curve]*, Second Curve*[Curve or real]*, Third Curve*[Curve or real]*, Output Curve (optional)*[Curve]*)
- Vec2d(First Curve*[Curve]*, Second Curve*[Curve or real]*, Output Curve (optional)*[Curve]*)
- Wif(First Curve*[Curve]*, Second Curve*[Curve]*)
- Window(Input Curve*[Curve]*, Window Type*[string]*, percentage lead in (optional)*[real]*, Output Curve (optional)*[Curve]*)
- Zero(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*)
- ZeroX(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*)
- ZeroY(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*)
- dB(Input Curve*[Curve]*, Reference Value*[real]*, Output Curve (optional)*[Curve]*)
- dBA(Input Curve*[Curve]*, Weighting Type*[String]*, Output Curve (optional)*[Curve]*)

Detailed Description

The Operate class allows you to use the built in curve operations in T/HIS to generate new curves. Most of the curve operations generate a new curve and return the curve object for the new curve. A few functions (NIJ, FFT, etc) generate multiple output curves and these return an array of curve objects.
See the documentation below for more details.

Details of functions

Abs(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*) [static]

Description

Convert a curve to absolute values

Arguments

- **Input Curve** (Curve)

Input Curve

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

To convert curve m to absolute values and store as curve p

```
p = Operate.Abs(m);
```

Acos(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*) [static]

Description

Calculate Arc Cosine

Arguments

- **Input Curve** (Curve)

Input Curve

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Calculate Arc Cosine() of curve m and store as curve p

```
p = Operate.Acos(m);
```

Acu(Input Curve*[Curve]*, Offset*[real]*, Time Period*[real]*, Output Curve (optional)*[Curve]*) [static]

Description

Evaluates the integratal of a curve over a user defined period

Arguments

- **Input Curve** (Curve)

Input Curve

- **Offset** (real)

User defined offset

- **Time Period** (real)

Time to integrate over

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Integrate c curve over 0.07 seconds with a 0.1 offset.

```
p = Operate.Acu(m,0.1,0.007);
```

Ad(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*) [static]

Description

Convert acceleration spectrum to a displacment spectrum

Arguments

- **Input Curve** (Curve)

Input Curve

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Convert curve m and store as curve p

```
p = Operate.Ad(m);
```

Add(Input Curve*[Curve]*, Second Curve or constant*[Curve or real]*, Output Curve (optional)*[Curve]*) [static]

Description

Add Y axis values

Arguments

- **Input Curve** (Curve)

Input Curve

- **Second Curve or constant** (Curve or real)

Second Curve or constant

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

To add curves m and n together and store as curve p

```
p = Operate.Add(m,n);
```

To add 20.0 to the values in curve m and store as curve p

```
p = Operate.Add(m,20.0);
```

Adx(First Curve*[Curve]*, Second Curve or constant*[Curve or real]*, Output Curve (optional)*[Curve]*) [static]

Description

Add X axis values

Arguments

- **First Curve** (Curve)

First Curve

- **Second Curve or constant** (Curve or real)

Second Curve or constant

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

To add X axis values for curves m and n together and store as curve p

```
p = Operate.Adx(m,n);
```

To add 20.0 to the X axis values in curve m and store as curve p

```
p = Operate.Adx(m,20.0);
```

Asi(X Acceleration*[Curve]*, Y Acceleration*[Curve]*, Z Acceleration*[Curve]*, Acceleration conversion factor*[real]*, X Acceleration Limit*[real]*, Y Acceleration Limit*[real]*, Z Acceleration Limit*[real]*, Calculation method*[string]*, X axis interval (optional)*[real]*, Output Curve (optional)*[Curve]*) [static]

Description

Acceleration Severity Index. This value is used to assess the performance of road side crash barriers. The calculation method can be set to 2010 (BS EN 1317-1:2010) or 1998 (BS EN 1317-1:1998).

Arguments

- **X Acceleration** (Curve)

X Acceleration Curve

- **Y Acceleration** (Curve)

Y Acceleration Curve

- **Z Acceleration** (Curve)

Z Acceleration Curve

- **Acceleration conversion factor** (real)

Factor required to divide input acceleration curve by to convert to (G)

- **X Acceleration Limit** (real)

X direction acceleration limit

- **Y Acceleration Limit** (real)

Y direction acceleration limit

- **Z Acceleration Limit** (real)

Z direction acceleration limit

- **Calculation method** (string)

Either 2010 or 1998.

- **X axis interval (optional)** (real)

If defined then T-HIS will automatically regularise the curve using this value first

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Calculate ASI using the 2010 method with input curves x,y and z, factors 12,9,10 and a conversioon factor of 9810. Regularise the input curves using an interval of 0.0001 first.

```
p = Operate.Asi(x,y,z,9810.0,12.0,9.0,10.0,"2010",0.0001);
```

Asin(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*) [static]

Description

Calculate Arc Sine

Arguments

- **Input Curve** (Curve)

Input Curve

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Calculate Arc Sine() of curve m and store as curve p

```
p = Operate.Asin(m);
```

Atan(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*) [static]

Description

Calculate Arc Tangent

Arguments

- **Input Curve** (Curve)

Input Curve

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Calculate Arc Tangent() of curve m and store as curve p

```
p = Operate.Atan(m);
```

Atan2(First Input Curve*[Curve]*, Second Input Curve*[Curve]*, Output Curve (optional)*[Curve]*) [static]

Description

Calculate Arc Tangent using atan2(y, x)

Arguments

- **First Input Curve** (Curve)

Input Curve

- **Second Input Curve** (Curve)

Input Curve

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Calculate Arc Tangent() of curve m / curve n and store as curve p

```
p = Operate.Atan2(m, n);
```

Av(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*) [static]

Description

Convert acceleration spectrum to a velocity spectrum

Arguments

- **Input Curve** (Curve)

Input Curve

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Convert curve m and store as curve p

```
p = Operate.Av(m);
```

Ave(Curves*[Array of Curve objects]*, Output Curve (optional)*[Curve]*) [static]

Description

Average a group of curves

Arguments

- **Curves** (Array of Curve objects)

Array of Curve objects

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Average the array of curves stored in curve array x and store as curve p

```
p = Operate.Ave(x);
```

Bes(Input Curve*[Curve]*, Frequency*[real]*, Order*[integer]*, X axis interval (optional)*[real]*, Output Curve (optional)*[Curve]*) [static]

Description

Bessel Filter

Arguments

- **Input Curve** (Curve)

Input Curve

- **Frequency** (real)

Cut-off Frequency (Hz)

- **Order** (integer)

Filter order

- **X axis interval (optional)** (real)

If defined then T-HIS will automatically regularise the curve using this value first

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Filter curve m using a cut-off of 400Hz and order 2 and output as curve p . Regularise the input curve using an interval of 0.0001 first.

```
p = Operate.Bes(m,400.0,2,0.0001);
```

Blc(Input Curve*[Curve]*) [static]

Description

Carry out a baseline correction on an accleration time history

Arguments

- **Input Curve** (Curve)

Moment / Time Curve

Returns

Array of Curve objects.
1st curve : Corrected curve
2nd curve : Integrated Velocity
3rd curve : Integrated Displacement

Return type

Array

Example

Calculate baseline correction on curve m, .

```
c_array = Operate.Blc(m);
corrected_curve = c_array[0];
vel_curve = c_array[1];
disp_curve = c_array[2];
```

But(Input Curve*[Curve]*, Frequency*[real]*, Order*[integer]*, X axis interval (optional)*[real]*, Output Curve (optional)*[Curve]*) [static]

Description

Butterworth Filter

Arguments

- **Input Curve** (Curve)

Input Curve

- **Frequency** (real)

Cut-off Frequency (Hz)

- **Order** (integer)

Filter order

- **X axis interval (optional)** (real)

If defined then T-HIS will automatically regularise the curve using this value first

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Filter curve m using a cut-off of 400Hz and order 2 and output as curve p . Regularise the input curve using an interval of 0.0001 first.

```
p = Operate.But(m,400.0,2,0.0001);
```

C1000(Input Curve*[Curve]*, X axis interval (optional)*[real]*, Output Curve (optional)*[Curve]*) [static]

Description

SAE Class 1000 Filter

Arguments

- **Input Curve** (Curve)

Input Curve

- **X axis interval (optional)** (real)

If defined then T-HIS will automatically regularise the curve using this value first

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Filter curve m and output as curve p . Regularise the input curve using an interval of 0.0001 first.

```
p = Operate.C1000(m,0.0001);
```

C180(Input Curve*[Curve]*, X axis interval (optional)*[real]*, Output Curve (optional)*[Curve]*) [static]

Description

SAE Class 180 Filter

Arguments

- **Input Curve** (Curve)

Input Curve

- **X axis interval (optional)** (real)

If defined then T-HIS will automatically regularise the curve using this value first

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Filter curve m and output as curve p . Regularise the input curve using an interval of 0.0001 first.

```
p = Operate.C180(m,0.0001);
```

C60(Input Curve*[Curve]*, X axis interval (optional)*[real]*, Output Curve (optional)*[Curve]*) [static]

Description

SAE Class 60 Filter

Arguments

- **Input Curve** (Curve)

Input Curve

- **X axis interval (optional)** (real)

If defined then T-HIS will automatically regularise the curve using this value first

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Filter curve m and output as curve p . Regularise the input curve using an interval of 0.0001 first.

```
p = Operate.C60(m,0.0001);
```

C600(Input Curve*[Curve]*, X axis interval (optional)*[real]*, Output Curve (optional)*[Curve]*) [static]

Description

SAE Class 600 Filter

Arguments

- **Input Curve** (Curve)

Input Curve

- **X axis interval (optional)** (real)

If defined then T-HIS will automatically regularise the curve using this value first

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Filter curve m and output as curve p . Regularise the input curve using an interval of 0.0001 first.

```
p = Operate.C600(m,0.0001);
```

Cat(First Curve*[Curve]*, Second Curve*[Curve or real]*, Output Curve (optional)*[Curve]*) [static]

Description

Concatenate 2 curves together

Arguments

- **First Curve** (Curve)

First Curve

- **Second Curve** (Curve or real)

Second Curve

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

To concatenate the values for curve n to those in curve m and store as curve p

```
p = Operate.Cat(m,n);
```

Clip(Input Curve*[Curve]*, X min*[real]*, X max*[real]*, Y min*[real]*, Y max*[real]*, Output Curve (optional)*[Curve]*) [static]

Description

Clip a curve

Arguments

- **Input Curve** (Curve)

Input Curve

- **X min** (real)

X minimum value

- **X max** (real)

X maximum value

- **Y min** (real)

Y minimum value

- **Y max** (real)

Y maximum value

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Clip a curve m to within 0.1<x<0.3, 0.0<y<100.0 and store as curve p

```
p = Operate.Clip(m,0.1,0.3,0.0,100.0);
```

Com(First Curve*[Curve]*, Second Curve*[Curve or real]*, Output Curve (optional)*[Curve]*) [static]

Description

Combine Y axis values from 2 curves together

Arguments

- **First Curve** (Curve)

First Curve

- **Second Curve** (Curve or real)

Second Curve

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

To combine the Y axis values for curve n to those in curve m and store as curve p

```
p = Operate.Com(m,n);
```

Cor(First Curve*[Curve]*, Second Curve*[Curve]*, Correlation type*[string]*) [static]

Description

Curve Correlation function. This Correlation function provides a measure of the degree to which two curves match. When comparing curves by eye, the quality of correlation may be judged on the basis of how well matched are the patterns of peaks, the overall shapes of the curves, etc, and can allow for differences of timing as well as magnitude. Thus a simple function based on the difference of Y-values (such as T/HIS ERR function) does not measure correlation in the same way as the human eye. The T/HIS correlation function attempts to include and quantify the more subtle ways in which the correlation of two curves may be judged.
The correlation can be calculated using either a strict or loose set of input parameters.
The degree of correlation is rated between 0 and 100.

Arguments

- **First Curve** (Curve)

First Curve

- **Second Curve** (Curve)

Second Curve

- **Correlation type** (string)

Correlation type, strict or loose

Returns

Correlation value

Return type

real

Example

Calculate the correlation between curves m and n using the strict input parameters.

```
val = Operate.Cor(m,n,"strict");
```

Cor3(First Curve*[Curve]*, Second Curve*[Curve]*, X axis factor (optional)*[real]*, Y axis factor (optional)*[real]*) [static]

Description

Curve Correlation function. This function first normalises the curves using two factors either specified by the user or defaults calculated by the program (the maximum absolute X and Y values of both graphs). For each point on the first normalised curve, the shortest distance to the second normalised curve is calculated. The root mean square value of all these distances is subtracted from 1 and then multiplied by 100 to get an index between 0 and 100. The process is repeated along the second curve and the two indices are averaged to get a final index. The higher the index the closer the correlation between the two curves.
Note that the choice of normalising factors is important. Incorrect factors may lead to a correlation index outside the range of 0 to 100

Arguments

- **First Curve** (Curve)

First Curve

- **Second Curve** (Curve)

Second Curve

- **X axis factor (optional)** (real)

Normalising factor used for X axis values

- **Y axis factor (optional)** (real)

Normalising factor used for Y axis values

Returns

Correlation value

Return type

real

Example

Calculate the correlation between curves m and n using the default normalising factors.

```
val = Operate.Cor3(m,n);
```

Calculate the correlation between curves m and n using 0.1 and 1000.0 as the X and Y normalising factors.

```
val = Operate.Cor3(m,n,0.1,1000);
```

Cos(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*) [static]

Description

Calculate Cosine

Arguments

- **Input Curve** (Curve)

Input Curve

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Calculate Cosine() of curve m and store as curve p

```
p = Operate.Cos(m);
```

Da(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*) [static]

Description

Convert displacment spectrum to an acceleration spectrum

Arguments

- **Input Curve** (Curve)

Input Curve

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Convert curve m and store as curve p

```
p = Operate.Da(m);
```

Dif(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*) [static]

Description

Differentiate a curve

Arguments

- **Input Curve** (Curve)

Input Curve

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

To differentiate curve m and store as curve p

```
p = Operate.Dif(m);
```

Div(First Curve*[Curve]*, Second Curve or constant*[Curve or real]*, Output Curve (optional)*[Curve]*) [static]

Description

Divide Y axis values

Arguments

- **First Curve** (Curve)

First Curve

- **Second Curve or constant** (Curve or real)

Second Curve or constant

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

To divide the Y axis values for curve n by curve m and store as curve p

```
p = Operate.Div(m,n);
```

To devide the Y axis values in curve m by 20.0 and store as curve p

```
p = Operate.Div(m,20.0);
```

Dix(First Curve*[Curve]*, Second Curve or constant*[Curve or real]*, Output Curve (optional)*[Curve]*) [static]

Description

Divide X axis values

Arguments

- **First Curve** (Curve)

First Curve

- **Second Curve or constant** (Curve or real)

Second Curve or constant

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

To divide the X axis values for curve n by curve m and store as curve p

```
p = Operate.Dix(m,n);
```

To devide the X axis values in curve m by 20.0 and store as curve p

```
p = Operate.Dix(m,20.0);
```

Dmg(Head Rotation Velocity X*[Curve]*, Head Rotation Velocity Y*[Curve]*, Head Rotation Velocity Z*[Curve]*, Calculation method*[string]*, X axis interval (optional)*[real]*, Filter Class (optional)*[string]*) [static]

Description

Damage Criterion DAMAGE Criterion is a brain injury metric which is based on deformation output from a second-order system of equation. DMG requires three input curves: Head Rotation Velocity X, Head Rotation Velocity Y, v. The function returns an array containing 4 curve objects. 1st Curve: Damage Resultant 2nd Curve: Damage X Component 3rd Curve: Damage Y Component 4th Curve: Damage Z Component

Arguments

- **Head Rotation Velocity X** (Curve)

Head Rotation Velocity X Curve

- **Head Rotation Velocity Y** (Curve)

Head Rotation Velocity Y Curve

- **Head Rotation Velocity Z** (Curve)

Head Rotation Velocity Z Curve

- **Calculation method** (string)

Calculation method used to solve Damage operation: ’rk4’, ’rkf45’, ’nbm’

- **X axis interval (optional)** (real)

If defined then T-HIS will automatically regularise the curve using this value first

- **Filter Class (optional)** (string)

If defined then T-HIS will automatically filter the input curve. The acceptable inputs for Filter class are ’C60’, ’C180’, ’C600’, ’C1000’

Returns

Array of Curve objects. 1st Curve: Damage Resultant 2nd Curve: Damage X Component 3rd Curve: Damage Y Component 4th Curve: Damage Z Component

Return type

Array

Example

Calculate DMG curves p using input head rotataional velocity curves m, n, o using rk4 method

```
p = Operate.Dmg(m,n,o, ’rk4’);
```

Calculate DMG curves p1 with additional arguments for 0.001 X axis interval and C60 Filter.

```
p1 = Operate.Dmg(m,n,o,’rk4’,0.0001,’C60’)
```

Ds(Input Curve*[Curve]*, Broadening Factor*[real]*, Redefine Frequencies*[string]*, Output Curve (optional)*[Curve]*) [static]

Description

Generate a design spectrum from a reponse spectrum

Arguments

- **Input Curve** (Curve)

Input Curve

- **Broadening Factor** (real)

Spectrum broadening factor

- **Redefine Frequencies** (string)

T-HIS selects a new set of frequencies for the output (yes or no)

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Convert curve m and let T-HIS determine the new frequencies, store as curve p

```
p = Operate.Ds(m,"yes");
```

Dv(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*) [static]

Description

Convert displacment spectrum to a velocity spectrum

Arguments

- **Input Curve** (Curve)

Input Curve

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Convert curve m and store as curve p

```
p = Operate.Dv(m);
```

Env(Curves*[Array of Curve objects]*, Output Curve (optional)*[Curve]*) [static]

Description

Generate an Envelope that bounds the min and max values of a group of curves

Arguments

- **Curves** (Array of Curve objects)

Array of Curve objects

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Envelope of curves stored in curve array x and store as curve p

```
p = Operate.Env(x);
```

Err(First Curve*[Curve]*, Second Curve*[Curve or real]*, Output Curve (optional)*[Curve]*) [static]

Description

Calculate the degree of correlation between 2 curves

Arguments

- **First Curve** (Curve)

First Curve

- **Second Curve** (Curve or real)

Second Curve

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

To calculate the correlation between curves n and m and store as curve p

```
p = Operate.Err(m,n);
```

Exc(Input Curve*[Curve]*, Output option*[string]*, Output Curve (optional)*[Curve]*) [static]

Description

Calculate and displays an EXCeedence plot. This is a plot of force (Y axis) versus cumulative time (X axis) for which the force level has been exceeded. By default the Automatic option will create an exceedence plot using either the +ve OR the -ve values depending on which the input curve contains most of.
The Positive option will calculate the exceedence plot using only the points with +ve y values.
The Negative option will calculate the exceedence plot using only the points with -ve y values.

Arguments

- **Input Curve** (Curve)

Input Curve

- **Output option** (string)

Select between automatic, positive or negative.

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Calculate Exceedence plot for curve m, using the positive option and store as curve p

```
p = Operate.Exc(m,"positive");
```

Exp(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*) [static]

Description

Calculate E to the power of Y axis values

Arguments

- **Input Curve** (Curve)

Input Curve

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Calculate E to the power of Y axis values for curve m and store as curve p

```
p = Operate.Exp(m);
```

Fft(Input Curve*[Curve]*, Output option*[string]*, X axis interval (optional)*[real]*, Scaling option (optional)*[string]*) [static]

Description

Fast Fourier Transform

Arguments

- **Input Curve** (Curve)

Input Curve

- **Output option** (string)

Generate magnitude, magnitude+phase or real+imaginary, (one of magnitude,phase,real)

- **X axis interval (optional)** (real)

If defined then T-HIS will automatically regularise the curve using this value first

- **Scaling option (optional)** (string)

Scaling option, (either one or two)

Returns

Curve object/array or NULL

Return type

Curve

Example

Generate magnitude and phase curves and return a curve array. Regularise the input curve using an interval of 0.0001 first and scale using option two.

```
c_array = Operate.Fft(m,"phase",0.0001,"one");
mag_curve = c_array[0];
phase_curve = c_array[1];
```

Fir(Input Curve*[Curve]*, X axis interval (optional)*[real]*, Output Curve (optional)*[Curve]*) [static]

Description

FIR Filter

Arguments

- **Input Curve** (Curve)

Input Curve

- **X axis interval (optional)** (real)

If defined then T-HIS will automatically regularise the curve using this value first

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Filter curve m and output as curve p . Regularise the input curve using an interval of 0.0001 first.

```
p = Operate.Fir(m,0.0001);
```

Hic(Input Curve*[Curve]*, Window*[real]*, Acceleration factor*[real]*) [static]

Description

HIC Calculation. After calculating the HIC value for a curve the value can also be obtained from the curve using the Curve.hic property. In addition to the HIC value the start and end time for the time window can also be obtained using the Curve.hic_tmin and Curve.hic_tmax properties.

Arguments

- **Input Curve** (Curve)

Input Curve

- **Window** (real)

Maximum time window

- **Acceleration factor** (real)

Factor required to divide input acceleration curve by to convert to (G)

Returns

HIC value

Return type

real

Example

Calculate HIC for curve m, using a window of 0.036s and a factor of 9810.

```
val = Operate.Hic(m,0.036,9810.0);
```

Hicd(Input Curve*[Curve]*, Window*[real]*, Acceleration factor*[real]*) [static]

Description

Modified HIC(d) Calculation for free motion headform. After calculating the HIC value for a curve the value can also be obtained from the curve using the Curve.hicd property. In addition to the HIC(d) value the start and end time for the time window can also be obtained using the Curve.hicd_tmin and Curve.hicd_tmax properties.

Arguments

- **Input Curve** (Curve)

Input Curve

- **Window** (real)

Maximum time window

- **Acceleration factor** (real)

Factor required to divide input acceleration curve by to convert to (G)

Returns

HIC(d) value

Return type

real

Example

Calculate HIC(d) for curve m, using a window of 0.036s and a factor of 9810.

```
val = Operate.Hicd(m,0.036,9810.0);
```

Ifft(First Curve*[Curve]*, Second Curve*[Curve]*, Input type*[string]*) [static]

Description

Inverse Fast Fourier Transform

Arguments

- **First Curve** (Curve)

First Curve

- **Second Curve** (Curve)

Second Curve

- **Input type** (string)

Specifies if inputs are magnitude+phase or real+imaginary, (magnitude or real)

Returns

Curve object or NULL

Return type

Curve

Example

Generate curve from magnitude (m) and phase (p) data and return as curve q.

```
q = Operate.Ifft(m,p,"magnitude");
```

Int(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*) [static]

Description

Integrate a curve

Arguments

- **Input Curve** (Curve)

Input Curve

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

To integrate curve m and store as curve p

```
p = Operate.Int(m);
```

Log(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*) [static]

Description

Calculate Natural Log of Y axis values

Arguments

- **Input Curve** (Curve)

Input Curve

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Calculate Natural Log of Y axis values for curve m and store as curve p

```
p = Operate.Log(m);
```

Log10(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*) [static]

Description

Calculate Log (base 10) of Y axis values

Arguments

- **Input Curve** (Curve)

Input Curve

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Calculate Log (base 10) of Y axis values for curve m and store as curve p

```
p = Operate.Log10(m);
```

Log10x(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*) [static]

Description

Calculate Log (base 10) of X axis values

Arguments

- **Input Curve** (Curve)

Input Curve

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Calculate Log (base 10) of X axis values for curve m and store as curve p

```
p = Operate.Log10x(m);
```

Logx(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*) [static]

Description

Calculate Natural Log of X axis values

Arguments

- **Input Curve** (Curve)

Input Curve

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Calculate Natural Log of X axis values for curve m and store as curve p

```
p = Operate.Logx(m);
```

Lsq(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*) [static]

Description

Calculate Least Squares Fit for a curve

Arguments

- **Input Curve** (Curve)

Input Curve

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

To calculate Least Squares Fit for curve m and store as curve p

```
p = Operate.Lsq(m);
```

Map(First Curve*[Curve]*, Second Curve*[Curve or real]*, Output Curve (optional)*[Curve]*) [static]

Description

Map Y axis values from one curve onto another curve

Arguments

- **First Curve** (Curve)

First Curve

- **Second Curve** (Curve or real)

Second Curve

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

To map curve n onto curve m and store as curve p

```
p = Operate.Map(m,n);
```

Max(Curves*[Array of Curve objects]*, Output Curve (optional)*[Curve]*) [static]

Description

Maximum of a group of curves

Arguments

- **Curves** (Array of Curve objects)

Array of Curve objects

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Maximum of curves stored in curve array x

```
p = Operate.Max(x);
```

Min(Curves*[Array of Curve objects]*, Output Curve (optional)*[Curve]*) [static]

Description

Minimum of a group of curves

Arguments

- **Curves** (Array of Curve objects)

Array of Curve objects

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Minimum of curves stored in curve array x

```
p = Operate.Min(x);
```

Mon(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*) [static]

Description

Sort a curve into monotonically increasing X axis values.

Arguments

- **Input Curve** (Curve)

Input Curve

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

To sort curve m and store as curve p

```
p = Operate.Mon(m);
```

Mul(First Curve*[Curve]*, Second Curve or constant*[Curve or real]*, Output Curve (optional)*[Curve]*) [static]

Description

Multiply Y axis values

Arguments

- **First Curve** (Curve)

First Curve

- **Second Curve or constant** (Curve or real)

Second Curve or constant

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

To multiply the Y axis values for curve n from m and store as curve p

```
p = Operate.Mul(m,n);
```

To multiply the Y axis values in curve m by 20.0 and store as curve p

```
p = Operate.Mul(m,20.0);
```

Mux(First Curve*[Curve]*, Second Curve or constant*[Curve or real]*, Output Curve (optional)*[Curve]*) [static]

Description

Multiply X axis values

Arguments

- **First Curve** (Curve)

First Curve

- **Second Curve or constant** (Curve or real)

Second Curve or constant

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

To multiply the X axis values for curve n from m and store as curve p

```
p = Operate.Mux(m,n);
```

To multiply the X axis values in curve m by 20.0 and store as curve p

```
p = Operate.Mux(m,20.0);
```

Ncp(First Curve*[Curve]*, Second Curve*[Curve]*) [static]

Description

Calculate a platic rotation curve for a beam from a moment/time and rotation/time

Arguments

- **First Curve** (Curve)

Moment / Time Curve

- **Second Curve** (Curve)

Rotation /Time Curve

Returns

Curve object or NULL

Return type

Curve

Example

Calculate plastic rotation curve p using curves m and r.

```
q = Operate.Ncp(m,r);
```

Nij(Shear Force*[Curve]*, Axial Force*[Curve]*, Moment*[Curve]*, Fzc_t*[real]*, Fzc_c*[real]*, Myc_f*[real]*, Myc_e*[real]*, E*[real]*) [static]

Description

Biomechanical neck injury predictor. Used as a measure of injury due to the load transferred through the occipital condyles.
This function returns an array containing 4 curve objects.
Curve 1 - "Nte" is the tension-extension condition
Curve 2 - "Ntf" is the tension-flexion condition
Curve 3 - "Nce" is the compression-extension condition
Curve 4 - "Ncf" is the compression-flexion condition.

Arguments

- **Shear Force** (Curve)

Shear Force Curve

- **Axial Force** (Curve)

Axial Force Curve

- **Moment** (Curve)

Moment Curve

- **Fzc_t** (real)

Critical Axial Force (Tension)

- **Fzc_c** (real)

Critical Axial Force (Compression)

- **Myc_f** (real)

Critical bending moment (Flexion)

- **Myc_e** (real)

Critical bending moment (Extension)

- **E** (real)

Distance

Returns

Array of Curve objects.
1st curve : Nte curve
2nd curve : Ntf curve
3rd curve : Nce curve
4th curve : Ncf curve

Return type

Array

Example

Calculate NIJ curves using input curves x,y,z, and constants Fzc=1.0 (tension) / 2.0 (compression), Myc=3.0 (flexion) / 4.0 (extenstion) and E=0.0.

```
c_array = Operate.Nij(x,y,z,1.0,2.0,3.0,4.0,0.0);
```

Nor(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*) [static]

Description

Normalise Y axis values between [-1,1]

Arguments

- **Input Curve** (Curve)

Input Curve

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Normalise Y axis values of curve m and store as curve p

```
p = Operate.Nor(m);
```

Nor2(Input Curve*[Curve]*, Y Min Value*[real]*, Y Max Value*[real]*, Lock to Axis Y Min*[integer]*, Lock to Axis Y Max*[integer]*, Output Curve (optional)*[Curve]*) [static]

Description

Normalise Y axis values with manual settings. The operation takes the absolute value of the user-specified Y Min and Y Max. It then finds the maximum of these two numbers and divides all Y data by this number. There are two locks which probe or "lock on to" the Y Max and Y Min axis values which offers quick axis-normalizing.

Arguments

- **Input Curve** (Curve)

First Curve

- **Y Min Value** (real)

The Minimum Y value

- **Y Max Value** (real)

The Maximum Y value

- **Lock to Axis Y Min** (integer)

Set the Lock button for the Y Minimum textbox

- **Lock to Axis Y Max** (integer)

Set the Lock button for the Y Maximum textbox

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Normalise the Y axis values of curve m taking the absolute maximum between the two values -200 and 100 (which for this example will equate to 200) with the Y Min Lock active and the Y Max Lock Inactive. This is then stored as curve p.

```
p = Operate.Nor2(m, -200, 100, 1, 0);
```

Nox(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*) [static]

Description

Normalise X axis values between [-1,1]

Arguments

- **Input Curve** (Curve)

Input Curve

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Normalise X axis values of curve m and store as curve p

```
p = Operate.Nox(m);
```

Nox2(Input Curve*[Curve]*, X Min Value*[real]*, X Max Value*[real]*, Lock to Axis X Min*[integer]*, Lock to Axis X Max*[integer]*, Output Curve (optional)*[Curve]*) [static]

Description

Normalise X axis values with manual settings. The operation takes the absolute value of the user-specified X Min and X Max. It then finds the maximum of these two numbers and divides all X data by this number. There are two locks which probe or "lock on to" the X Max and X Min axis values which offers quick axis-normalizing.

Arguments

- **Input Curve** (Curve)

First Curve

- **X Min Value** (real)

The Minimum X value

- **X Max Value** (real)

The Maximum X value

- **Lock to Axis X Min** (integer)

Set the Lock button for the X Minimum textbox

- **Lock to Axis X Max** (integer)

Set the Lock button for the X Maximum textbox

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Normalise the X axis values of curve m taking the absolute maximum between the two values -200 and 100 (which for this example will equate to 200) with the X Min Lock active and the X Max Lock Inactive. This is then stored as curve p.

```
p = Operate.Nox2(m, -200, 100, 1, 0);
```

Octave(Input Curve*[Curve]*, Band type to convert to*[String]*, Output Type*[String]*, Input Type*[String]*, Output Curve (optional)*[Curve]*) [static]

Description

Coverts a narrow band curve to either Octave or 1/Third Octave bands

Arguments

- **Input Curve** (Curve)

Input Curve

- **Band type to convert to** (String)

Band type to convert to. Either "Octave" or "Third" Octave.

- **Output Type** (String)

Generate curve containing either "RMS" or "mean" values.

- **Input Type** (String)

Input curve contains either "Linear" or "dB" values.

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Convert curve m that contains Linear values to 1/3 Octave bands and output RMS in curve p

```
p = Operate.Octave(m,"third","rms","linear");
```

Olc(Input Curve*[Curve]*, Second Curve or constant*[Curve or real]*, X axis interval (optional)*[real]*, Filter Class (optional)*[string]*) [static]

Description

Occupant load Criterion. Used as a parameter to evaluate Euro NCAP MPDB assessment as specified in Technical Bulletin TB 027 v1.1.1, which is intended to be used with the Adult Occupant Protection. The function returns an array containing 5 curve objects. Curve 1 - Velocity of Virtual Occupant Curve 2 - Velocity of the Barrier Model Curve 3 - Displacement of the Barrier Model Curve 4 - Displacement of the Virtual Occupant Curve 5 - Relative Displacement between the two models

Arguments

- **Input Curve** (Curve)

Input Curve

- **Second Curve or constant** (Curve or real)

Second Curve or constant

- **X axis interval (optional)** (real)

If defined then T-HIS will automatically regularise the curve using this value first

- **Filter Class (optional)** (string)

If defined then T-HIS will automatically filter the input curve. The acceptable inputs for Filter class are ’C60’, ’C180’, ’C600’, ’C1000’

Returns

Array of Curve objects. 1st Curve: Velocity of Virtual Occupant 2nd Curve: Velocity of the Barrier Model 3rd Curve: Displacement of the Barrier Model 4th Curve: Displacement of the Virtual Occupant 5th Curve: Relative Displacement between the two models

Return type

Array

Example

Calculate OLC curves p using input acceleration curve m and velocity curve n

```
p = Operate.Olc(m,n);
```

Calculate OLC curves p using input acceleration m curve and Initial Velocity Value 13888.0

```
p = Operate.Olc(m,13888.0);
```

Calculate OLC curves p1 and p2 with additional arguments for 0.001 X axis interval and C60 Filter.

```
p1 = Operate.Olc(m,n,0.0001,'C60')

p2 = Operate.Olc(m,13888.0,0.0001,'C60')
```

Order(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*) [static]

Description

Reverse the order of points in a curve

Arguments

- **Input Curve** (Curve)

Input Curve

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Reverse the order of points in curve m and store as curve p

```
p = Operate.Order(m);
```

Pbut(Input Curve*[Curve]*, Frequency*[real]*, Order*[integer]*, X axis interval (optional)*[real]*, Output Curve (optional)*[Curve]*) [static]

Description

Pure Butterworth Filter

Arguments

- **Input Curve** (Curve)

Input Curve

- **Frequency** (real)

Cut-off Frequency (Hz)

- **Order** (integer)

Filter order

- **X axis interval (optional)** (real)

If defined then T-HIS will automatically regularise the curve using this value first

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Filter curve m using a cut-off of 400Hz and order 2 and output as curve p. Regularise the input curve using an interval of 0.0001 first.

```
p = Operate.Pbut(m,400.0,2,0.0001);
```

Power(Input Curve*[Curve]*, Power*[real]*, Output Curve (optional)*[Curve]*) [static]

Description

Raise to the power

Arguments

- **Input Curve** (Curve)

Input Curve

- **Power** (real)

Power to raise Y axis values by

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Raise the Y axis values for curve m to the power 2.5 and store as curve p

```
p = Operate.Power(m,2.5);
```

Rave(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*) [static]

Description

Calculate rolling average of a curve

Arguments

- **Input Curve** (Curve)

Input Curve

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Calculate rolling average of curve m and store as curve p

```
p = Operate.Rave(m);
```

Rec(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*) [static]

Description

Calculate reciprocal

Arguments

- **Input Curve** (Curve)

Input Curve

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Calculate receprocal of curve m and store as curve p

```
p = Operate.Rec(m);
```

Reg(Input Curve*[Curve]*, X axis interval*[real]*, Output Curve (optional)*[Curve]*) [static]

Description

Regularise X axis intervals for a curve.

Arguments

- **Input Curve** (Curve)

Input Curve

- **X axis interval** (real)

New X axis interval

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Regularise curve m using a new X axis intreval of 0.0001.

```
p = Operate.Reg(m,0.0001);
```

Res(Curves*[Array of Curve objects]*, Output Curve (optional)*[Curve]*) [static]

Description

Resultant of a group of curves

Arguments

- **Curves** (Array of Curve objects)

Array of Curve objects

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Resultant of curves stored in curve array x

```
p = Operate.Res(x);
```

Rev(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*) [static]

Description

Reverse X and Y axis values

Arguments

- **Input Curve** (Curve)

Input Curve

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Reverse X and Y axis values of curve m and store as curve p

```
p = Operate.Rev(m);
```

Rs(Input Curve*[Curve]*, Damping Factor*[real]*, Sampling Points*[int]*, X axis interval (optional)*[real]*, Output Curve (optional)*[Curve]*) [static]

Description

Generate a reponse spectrum from input accelerations

Arguments

- **Input Curve** (Curve)

Input Curve

- **Damping Factor** (real)

Dammping factor

- **Sampling Points** (int)

Number of points to sample over (30 or 70)

- **X axis interval (optional)** (real)

If defined then T-HIS will automatically regularise the curve using this value first

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Array of Curve objects
1st curve : Relative displacement
2nd curve : Relative velocity
3th curve : Pseudo relative velocity
4th curve : Absolute acceleration
5th curve : Pseudo absolute acceleration

Return type

Array

Example

Generate a response spectrum using a factor of 0.05 and 70 sampling points. Regularise the input curve using an interval of 0.0001 first.

```
p = Operate.Rs(m,0.05,70,0.0001);
```

Sin(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*) [static]

Description

Calculate Sine

Arguments

- **Input Curve** (Curve)

Input Curve

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Calculate Sine() of curve m and store as curve p

```
p = Operate.Sin(m);
```

Smooth(Input Curve*[Curve]*, Smoothing Factor*[integer]*, Output Curve (optional)*[Curve]*) [static]

Description

Apply a smoothing factor to a curve

Arguments

- **Input Curve** (Curve)

Input Curve

- **Smoothing Factor** (integer)

Number of points to average over

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Smooth curve m using 7 points and store as curve p

```
p = Operate.Smooth(m,7);
```

Sqr(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*) [static]

Description

Square root of a curve

Arguments

- **Input Curve** (Curve)

Input Curve

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Square root curve m and store as curve p

```
p = Operate.Sqr(m);
```

Stress(Input Curve*[Curve]*, Convert to*[string]*, Output Curve (optional)*[Curve]*) [static]

Description

Convert between true and engineering stress

Arguments

- **Input Curve** (Curve)

Input Curve

- **Convert to** (string)

Type to convert to (True or Engineering)

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Convert curve m from engineering to true strees and store as curve p

```
p = Operate.Stress(m,"True");
```

Sub(First Curve*[Curve]*, Second Curve or constant*[Curve or real]*, Output Curve (optional)*[Curve]*) [static]

Description

Subtract Y axis values

Arguments

- **First Curve** (Curve)

First Curve

- **Second Curve or constant** (Curve or real)

Second Curve or constant

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

To subtract the Y axis values for curve n from m and store as curve p

```
p = Operate.Sub(m,n);
```

To subtract 20.0 from the Y axis values in curve m and store as curve p

```
p = Operate.Sub(m,20.0);
```

Sum(Curves*[Array of Curve objects]*, Output Curve (optional)*[Curve]*) [static]

Description

Sum of a group of curves

Arguments

- **Curves** (Array of Curve objects)

Array of Curve objects

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Sum of curves stored in curve array x

```
p = Operate.Sum(x);
```

Sux(First Curve*[Curve]*, Second Curve or constant*[Curve or real]*, Output Curve (optional)*[Curve]*) [static]

Description

Subtract X axis values

Arguments

- **First Curve** (Curve)

First Curve

- **Second Curve or constant** (Curve or real)

Second Curve or constant

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

To subtract the X axis values for curve n from m and store as curve p

```
p = Operate.Sux(m,n);
```

To subtract 20.0 from the X axis values in curve m and store as curve p

```
p = Operate.Sux(m,20.0);
```

Tan(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*) [static]

Description

Calculate Tangent

Arguments

- **Input Curve** (Curve)

Input Curve

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Calculate Tangent() of curve m and store as curve p

```
p = Operate.Tan(m);
```

Thiv(X Acceleration*[Curve]*, Y Acceleration*[Curve]*, Yaw Rate*[Curve]*, Dx*[real]*, Dy*[real]*, X0*[real]*) [static]

Description

Theoretical Head Impact Velocity and the Post Impact Head Deceleration. These values are used to assess the performance of road side crash barriers.
This function returns an array containing 2 curve objects. The 1st curve is the THIV curve and the 2nd is the PHD curve. The peak values of these curves are the corresponding THIV and PHD values and can be obtained using the Curve.ymax property.

Arguments

- **X Acceleration** (Curve)

X Acceleration Curve

- **Y Acceleration** (Curve)

Y Acceleration Curve

- **Yaw Rate** (Curve)

Yaw Rate Curve

- **Dx** (real)

Horizontal distance between occupants head and vehicle

- **Dy** (real)

Lateral distance between occupants head and vehicle

- **X0** (real)

Horizontal distance between occupants head and vehicle CofG

Returns

Array of Curve objects.
1st curve : THIV curve
2nd curve : PHD curve

Return type

Array

Example

Calculate THIV and PHD curves x,y,z and distances Dx=0.6, Dy=0.3, X0=0.0.

```
c_array = Operate.Thiv(x,y,z,0.6,0.3,0.0);
thiv = c_array[0].ymax;
phd = c_array[1].ymax;
```

Ti(Axial Force*[Curve]*, X Moment*[Curve]*, Y Moment*[Curve]*, Fzc*[real]*, Mrc*[real]*, X axis interval (optional)*[real]*, Filter Class (optional)*[string]*, Output Curve (optional)*[Curve]*) [static]

Description

Tibia Index is an injury criterion for the lower leg area used to predict leg injuries.

Arguments

- **Axial Force** (Curve)

Axial Force Curve

- **X Moment** (Curve)

X Moment Curve

- **Y Moment** (Curve)

Y Moment Curve

- **Fzc** (real)

Critical Axial Force

- **Mrc** (real)

Critical Resultant Moment

- **X axis interval (optional)** (real)

If defined then T-HIS will automatically regularise the curve using this value first

- **Filter Class (optional)** (string)

If defined then T-HIS will automatically filter the input curve. The acceptable inputs for Filter class are ’C60’, ’C180’, ’C600’, ’C1000’

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Calculate TI Curve t using input curves x,y,z and constants Fzc=35.9 and Mrc=225.0

```
t = Operate.Ti(x,y,z,35.9,225.0);
```

Calculate TI curve t with additional arguments 0.0001 for X axis interval and C600 for Filter Class.

```
t = Operate.Ti(x,y,z,,35.9,225.0,0.0001,'C600')
```

Tms(Input Curve*[Curve]*, Period*[real]*) [static]

Description

3ms Clip Calculation. After calculating the 3ms clip value for a curve the value can also be obtained from the curve using the Curve.tms property. In addition to the 3ms clip value the start and end time for the time window can also be obtained using the Curve.tms_tmin and Curve.tms_tmax properties.

Arguments

- **Input Curve** (Curve)

Input Curve

- **Period** (real)

Clip period

Returns

3ms clip value

Return type

real

Example

Calculate 3ms clip for curve m, using a clip period of 0.003s.

```
val = Operate.Tms(m,0.003);
```

Translate(Input Curve*[Curve]*, X value*[real]*, Y value*[real]*, Output Curve (optional)*[Curve]*) [static]

Description

Translate a curve

Arguments

- **Input Curve** (Curve)

Input Curve

- **X value** (real)

X translation value

- **Y value** (real)

Y translation value

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Translate curve m by x=0.2, y=0.3 and store as curve p

```
p = Operate.Translate(m,0.2,0.3);
```

Tti(Upper Rib Acceleration*[Curve]*, Lower Rib Acceleration*[Curve]*, T12 Acceleration*[Curve]*) [static]

Description

Thorax Trauma Index.

Arguments

- **Upper Rib Acceleration** (Curve)

Upper Rib Acceleration Curve

- **Lower Rib Acceleration** (Curve)

Lower Rib Acceleration Curve

- **T12 Acceleration** (Curve)

T12 Acceleration Curve

Returns

TTI value

Return type

real

Example

Calculate TTI using curves x,y and z as inputs.

```
val = Operate.TTi(x,y,z);
```

Va(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*) [static]

Description

Convert velocity spectrum to an acceleration spectrum

Arguments

- **Input Curve** (Curve)

Input Curve

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Convert curve m and store as curve p

```
p = Operate.Va(m);
```

Vc(Input Curve*[Curve]*, A*[real]*, B*[real]*, Calculation method*[string]*, Output Curve (optional)*[Curve]*) [static]

Description

Viscous Criteria calculate. The VC calculation can be done using 2 different calculation methods ECER95 and IIHS.

Arguments

- **Input Curve** (Curve)

Input Curve

- **A** (real)

Constant A

- **B** (real)

Constant B

- **Calculation method** (string)

Either ECER95 or IIHS.

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Calculate VC for curve m, using A=1.3, B=0.229 and the ECER95 method

```
p = Operate.Vc(m,1.3,0.229,"ECER95");
```

Vd(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*) [static]

Description

Convert velocity spectrum to a displacment spectrum

Arguments

- **Input Curve** (Curve)

Input Curve

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Convert curve m and store as curve p

```
p = Operate.Vd(m);
```

Vec(First Curve*[Curve]*, Second Curve*[Curve or real]*, Third Curve*[Curve or real]*, Output Curve (optional)*[Curve]*) [static]

Description

Vector magnitude of 3 curves

Arguments

- **First Curve** (Curve)

First Curve

- **Second Curve** (Curve or real)

Second Curve

- **Third Curve** (Curve or real)

Second Curve

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Calculate vector magnitude of curves m,n,o and store as curve p

```
p = Operate.Vec(m,n,o);
```

Vec2d(First Curve*[Curve]*, Second Curve*[Curve or real]*, Output Curve (optional)*[Curve]*) [static]

Description

Vector magnitude of 2 curves

Arguments

- **First Curve** (Curve)

First Curve

- **Second Curve** (Curve or real)

Second Curve

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Calculate vector magnitude of curves m and n and store as curve p

```
p = Operate.Vec2d(m,n);
```

Wif(First Curve*[Curve]*, Second Curve*[Curve]*) [static]

Description

Weigthed Integrated Factor (WIFAC) Correlation function.

Arguments

- **First Curve** (Curve)

First Curve

- **Second Curve** (Curve)

Second Curve

Returns

Correlation value

Return type

real

Example

Calculate the correlation between curves m and n.

```
val = Operate.Wif(m,n);
```

Window(Input Curve*[Curve]*, Window Type*[string]*, percentage lead in (optional)*[real]*, Output Curve (optional)*[Curve]*) [static]

Description

Apply a smoothing window to a curve

Arguments

- **Input Curve** (Curve)

Input Curve

- **Window Type** (string)

Window type to apply (Hanning, cosine or exponetial)

- **percentage lead in (optional)** (real)

percentage lead in for cosine window

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Apply a hanning window to curve m and store as curve p

```
p = Operate.Window(m,"Hanning");
```

Zero(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*) [static]

Description

Translate curve to 0,0

Arguments

- **Input Curve** (Curve)

Input Curve

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Translate curve m to (0,0) and store as curve p

```
p = Operate.Zero(m);
```

ZeroX(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*) [static]

Description

Translate curve to X=0.0

Arguments

- **Input Curve** (Curve)

Input Curve

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Translate curve m to X=0 and store as curve p

```
p = Operate.ZeroX(m);
```

ZeroY(Input Curve*[Curve]*, Output Curve (optional)*[Curve]*) [static]

Description

Translate curve to Y=0.0

Arguments

- **Input Curve** (Curve)

Input Curve

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Translate curve m to Y=0 and store as curve p

```
p = Operate.ZeroY(m);
```

---

dB(Input Curve*[Curve]*, Reference Value*[real]*, Output Curve (optional)*[Curve]*) [static]

Description

Converts a curve to dB (y = 20.0*log(y/yref))

Arguments

- **Input Curve** (Curve)

Input Curve

- **Reference Value** (real)

Reference value

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Convert curve m to dB’s using a reference value of 10.0 and store as curve p

```
p = Operate.dB(m,10.0);
```

---

dBA(Input Curve*[Curve]*, Weighting Type*[String]*, Output Curve (optional)*[Curve]*) [static]

Description

Applies A-weighting to a curve (convert from dB to dBA)

Arguments

- **Input Curve** (Curve)

Input Curve

- **Weighting Type** (String)

Apply either Narrow band (narrow) or Octave band (octave) A weighting

- **Output Curve (optional)** (Curve)

Curve to overwrite

Returns

Curve object or NULL

Return type

Curve

Example

Apply narrow band A-weighting to convert curve m from dB to dBA and store as curve p

```
p = Operate.dBA(m,"narrow");
```

Options class

The Options class enables you to access several options in T/HIS. More...

The T/HIS JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Options class properties

Name	Type	Description
auto_ confirm	logical	If true then T/HIS will automatically confirm (i.e. press the OK button) on (most) message boxes that are mapped. If false (default) then the message boxes will be shown and wait for the user to press a button. This option may be useful to help automate an operation where T/HIS would normally show a message box and wait for the user to press a button.
exception_ messages	logical	If true (default) error messages will be printed to the dialogue box/stdout when an exception occurs in the API. If false they will not be printed. This option may be useful if you are using try/catch to manage exceptions and you do not want any error messages to be printed.

Properties for ssh

Name	Type	Description
ssh_ buffer_ size	integer	The size of the buffer used (in kiloBytes) when transferring data to/from the remote machine in the Ssh class. Depending on your network and the size of the files you are transferring, changing this value may make file transfers quicker. The default value is 64(kB) but any value in the range 1(kB) to 1024(kB) is allowed.

Properties for widgets

Name	Type	Description
max_widgets	integer	The maximum number of Widgets that can be made for one Window. The default value is 1000
max_ window_ lines	integer	The maximum number of lines that can be made for a Window.Error(), Window.Information(), Window.Message(), Window.Question() or Window.Warning() window. The default value is 25

Detailed Description

The Options class is used to get/set options that T/HIS uses for certain functions. The options are available as **class** properties. See the documentation for more details. An example: Options.mass_properties_include_attached_mass_ deformable_elems=true

Page class

The Page class allows you to return or set the current active page in T/HIS. More...

The T/HIS JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- AddGraph(Page number*[integer]*, Graph number (optional)*[integer]*, Graph number to copy properties from (optional)*[integer]*, Number of graphs (optional)*[integer]*)
- CreatePage()
- DeletePage(Page number*[integer]*)
- GetPageFromName(Page name*[integer]*)
- GetPageName(Page number*[integer]*)
- Layout(Page number*[integer]*, Layout*[String or Integer]*, Num in X (optional)*[integer]*, Num in Y (optional)*[integer]*)
- MaxAllowable()
- RemoveGraph(Page number*[integer]*, Graph number (optional)*[integer]*, Lower end of range for removing graphs (optional)*[integer]*, Upper end of range for removing graphs (optional)*[integer]*)
- ReturnActivePage()
- ReturnGraphs(Page number*[integer]*)
- SetActivePage(Page number*[integer]*)
- SetPageName(Page number*[integer]*, Page name*[string]*)
- Total()

Detailed Description

The Page class allows you to return or set the current active page in T/HIS.

Details of functions

AddGraph(Page number*[integer]*, Graph number (optional)*[integer]*, Graph number to copy properties from (optional)*[integer]*, Number of graphs (optional)*[integer]*) [static]

Description

Adds one or more graphs to the specified page.

Arguments

- **Page number** (integer)

Page number to add graph(s) to.

- **Graph number (optional)** (integer)

Graph number to add to page. If this argument is 0 or not given, a new graph is created.

- **Graph number to copy properties from (optional)** (integer)

If the second argument is 0, this specifies which graph to copy properties from when creating new graphs.

- **Number of graphs (optional)** (integer)

If the second argument is 0, this specifies the number of new graphs to create and add to the specified page.

Returns

True if the graph was added, false if failed.

Return type

Boolean

Example

To add graph 1 to page 2

```
Page.AddGraph(2,1);
```

CreatePage() [static]

Description

Creates Page

Arguments

No arguments

Returns

Boolean

Return type

Boolean

Example

Add new page.

```
Page.CreatePage();
```

DeletePage(Page number*[integer]*) [static]

Description

Deletes Page

Arguments

- **Page number** (integer)

The page number which needs to be deleted.

Returns

String

Return type

String

Example

Delete page 2.

```
Page.DeletePage(2);
```

GetPageFromName(Page name*[integer]*) [static]

Description

Retrieves the id of a page from it’s name.

Arguments

- **Page name** (integer)

The page name for which the page id will be retrieved.

Returns

Integer

Return type

Number

Example

Retrieve the id of page name.

```
Page.GetPageFromName("name");
```

GetPageName(Page number*[integer]*) [static]

Description

Retrieves the name of a page.

Arguments

- **Page number** (integer)

The page number for which the name will be retrieved.

Returns

String

Return type

String

Example

Retrieve the name of page 2.

```
Page.GetPageName(2);
```

Layout(Page number*[integer]*, Layout*[String or Integer]*, Num in X (optional)*[integer]*, Num in Y (optional)*[integer]*) [static]

Description

Sets the layout of either all pages or a specified page.

Arguments

- **Page number** (integer)

Page number for which to set layout. If this argument is 0 then layout will be set on all pages individually. If -1 then the layout will be set globally, as in the ’Graphs’ panel.

- **Layout** (String or Integer)

Layout specifier. Options are: "wide" or 1 - Tile wide, "tall" or 2 - Tile tall, "1x1" or 3 - 1x1, "2x2" or 4 - 2x2, "3x3" or 5 - 3x3, "xy" or 6 - XxY.

- **Num in X (optional)** (integer)

Number of graphs in X-direction if user-defined XxY layout (6).

- **Num in Y (optional)** (integer)

Number of graphs in Y-direction if user-defined XxY layout (6).

Returns

True if the layout was set, false if failed.

Return type

Boolean

Example

To set the layout of page 1 to 'Tile tall'.

```
Page.Layout(1,"tall");
```

MaxAllowable() [static]

Description

Returns the maximum number of pages allowed. Useful because this differs between linked and unlinked T/HIS and can be used to check before trying to reference a page number that may be out-of-range.

Arguments

No arguments

Returns

Integer

Return type

Number

Example

To find the maximum allowable number of pages in T/HIS:

```
var maxPages = Page.MaxAllowable();
```

RemoveGraph(Page number*[integer]*, Graph number (optional)*[integer]*, Lower end of range for removing graphs (optional)*[integer]*, Upper end of range for removing graphs (optional)*[integer]*) [static]

Description

Remove one or more graphs from the specified page.

Arguments

- **Page number** (integer)

Page number to remove the graph from.

- **Graph number (optional)** (integer)

Graph number to remove from page. If this argument is 0 or not given, the highest number graph on the page will be removed. If this argument is -1, all graphs will be removed.

- **Lower end of range for removing graphs (optional)** (integer)

If the second argument is 0, this specifies the lower end of the range for removing graphs. All graphs with numbers within the specified range will be removed from the page.

- **Upper end of range for removing graphs (optional)** (integer)

If the second argument is 0, this specifies the upper end of the range for removing graphs. All graphs with numbers within the specified range will be removed from the page. If this argument is not given then it will be set to the maximum number of graphs by default.

Returns

True if the graph was removed, false if failed.

Return type

Boolean

Example

To remove any graph with number between 2 and 6 from page 3

```
Page.RemoveGraph(3,0,2,6);
```

ReturnActivePage() [static]

Description

Returns the current active page in T/HIS.

Arguments

No arguments

Returns

integer

Return type

Number

Example

To return the current active page

```
var p = Page.ReturnActivePage();
```

ReturnGraphs(Page number*[integer]*) [static]

Description

Returns the graphs on the specified page as an array of Graph objects.

Arguments

- **Page number** (integer)

Page number for which to return the graphs it contains.

Returns

Array of Graph objects

Return type

Array

Example

To return the graphs on page 2.

```
Page.ReturnGraphs(2);
```

SetActivePage(Page number*[integer]*) [static]

Description

Sets the current active page in T/HIS, returning false if the page does not exist.

Arguments

- **Page number** (integer)

Page number to set to active page

Returns

True if the page was set, false if the page does not exist.

Return type

Boolean

Example

To set the current active page to 2

```
Page.SetActivePage(2);
```

SetPageName(Page number*[integer]*, Page name*[string]*) [static]

Description

Sets the Name of a Page

Arguments

- **Page number** (integer)

The page number for which the name will be set.

- **Page name** (string)

The name to assign to the page.

Returns

Boolean

Return type

Boolean

Example

To remove name on page 2.

```
Page.SetPageName(2,"");
```

Total() [static]

Description

Returns the total number of pages.

Arguments

No arguments

Returns

integer

Return type

Number

Example

To find how many pages there are in T/HIS:

```
var num = Page.Total();
```

PopupWindow class

The PopupWindow class allows you to create popup windows for a graphical user interface. More...

The T/HIS JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Member functions

- Hide()

PopupWindow properties

Name	Type	Description
persistent	boolean	If the popup window will remain mapped when a button is pressed in it. By default (false) when a button is pressed in the popup window the popup will be unmapped. If set to true then the popup will remain mapped until the user clicks out of the window or hides it by calling Hide()

Detailed Description

The PopupWindow class allows you to make popup windows (that you can place Widgets in) and link them to Widgets. The popup window is then displayed by right clicking on the Widget the popup is linked to. The following very simple example shows how to create a popup window and link it to a label Widget.

```
// Create popup window
var pw = new PopupWindow();
// Create some widgets in the popup window
var pl = new Widget(pw, Widget.LABEL,   1, 30,  1,  7, "Label");
var pb = new Widget(pw, Widget.BUTTON,  1, 30,  7, 13, "Button");
var pt = new Widget(pw, Widget.TEXTBOX, 1, 30, 20, 26, "Textbox");
// Create window with title "Popup example" from 0.8-1.0 in x and 0.5-0.6 in y
var w = new Window("Popup example", 0.8, 1.0, 0.5, 0.6);
// Create label widget
var l = new Widget(w, Widget.LABEL,   1, 50, 1,  7, "Right click for popup...");
// link popup window to widget
l.popupWindow = pw;
// Assign the onPopup callback method to the function 'do_popup'
// This is only required if you want to make any changes before the popup
appears
l.onPopup  = do_popup;
// Show the widget and start event loop
w.Show();
////////////////////////////////////////////////////////////////////////////
function do_popup()
{
    Message("Showing popup");
}
```

See the documentation below and the Widget class for more details.

Constructor

new PopupWindow()

Description

Create a new PopupWindow object.

Arguments

No arguments

Returns

PopupWindow object

Return type

PopupWindow

Example

To create a PopupWindow containing the buttons "Create" and "Edit" and link it to button b:

```
var pw = new PopupWindow();
var c = new Widget(pw, Widget.BUTTON, 1, 30, 1, 7,  "Create");
var e = new Widget(pw, Widget.BUTTON, 1, 30, 7, 13, "Edit");
b.popupWindow = pw;
```

Details of functions

Hide()

Description

Hides (unmaps) the popup window.

Arguments

No arguments

Returns

No return value

Example

To hide popup window w:

```
w.Hide();
```

Read class

The Read class allows the user to read CSV, Curve and other filetypes into T/HIS. More...

The T/HIS JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Bulk(Filename*[String]*, Options (optional)*[object]*)
- CSV(Filename*[String]*, Options (optional)*[object]*)
- Cur(Filename*[String]*, Options (optional)*[object]*)
- DIAdem(Filename*[String]*, X-axis channel*[integer]*, Options (optional)*[object]*)
- Equation(Formula*[String]*, Options (optional)*[object]*)
- ISO(Filename*[String]*, Options (optional)*[object]*)
- Key(Filename*[String]*, Options (optional)*[object]*)
- LSPP(Filename*[String]*, Options (optional)*[object]*)

Read constants

Name	Description
Read.CSV_COMMA	CSV comma separator
Read.CSV_SPACE	CSV space separator
Read.CSV_TAB	CSV tab separator
Read.CSV_XYXY	CSV format X,Y,X,Y
Read.CSV_XYYY	CSV format X,Y,Y,Y
Read.DIADEM_COMMENT	Diadem comment written to curve tag
Read.DIADEM_NAME	Diadem channel name written to curve tag
Read.EQUATION_CURVE_VARS	Calculate x values from curve variables for equations
Read.EQUATION_X_OR_CURVE	If there are no curve variables, use the X start, end and interval values for equation x values. Otherwise calculate x values from the curve variables.
Read.EQUATION_X_VALS	Use the X start, end and interval values for equation x values
Read.ISO_CHANNEL_CODE	Use the channel code for the ISO curve labels
Read.ISO_CHANNEL_LABEL	Use the channel label for the ISO curve labels
Read.ISO_MULTIPLE_CHANNELS	Multiple channels ISO file
Read.ISO_SINGLE_CHANNEL	Single channel ISO file
Read.LSPP_CURVE_FILE	LSPP curve file format
Read.LSPP_XY_PAIRS	LSPP XY pairs file format

Detailed Description

The Read class allows the user to read CSV, Curve and other filetypes into T/HIS.

Details of functions

Bulk(Filename*[String]*, Options (optional)*[object]*) [static]

Description

Reads a Bulk Data file into T/HIS.

Arguments

- **Filename** (String)

Name of Bulk Data file to read

- **Options (optional)** (object)

Options which give you greater control of reading a Bulk file:

Object has the following properties:

Name	Type	Description
outputOpt (optional)	integer/string	Allows you to control which curve ID number the file will begin to read at. This can either be a whole number greater than 0 or specified as a string like "#3" for curve 3. There are also the special string characters "%", standing for highest free curve as well as "#", standing for the lowest free curve. By default, this will be set to the highest free curve.

Returns

No return value

Example

To read a Bulk Data file named "example.bdf"

```
Read.Bulk("example.bdf");
```

To read a Bulk Data file named "example.bdf" at the lowest free curve

```
var options = new Object();
options.outputOpt = "#";
Read.Bulk("example.bdf",options);
```

CSV(Filename*[String]*, Options (optional)*[object]*) [static]

Description

Reads a CSV file into T/HIS.

Arguments

- **Filename** (String)

Name of CSV file to read.

- **Options (optional)** (object)

Options which give you greater control of reading a CSV file:

Object has the following properties:

Name	Type	Description
csvType (optional)	integer	CSV file type. Can be Read.CSV_XYXY or Read.CSV_XYYY

outputOpt (optional)	integer/string	Allows you to control which curve ID number the file will begin to read at. This can either be a whole number greater than 0 or specified as a string like "#3" for curve 3. There are also the special string characters "%", standing for highest free curve as well as "#", standing for the lowest free curve. By default, this will be set to the highest free curve.
rowAxisLabel (optional)	integer	Index of the row containing axis labels. This is row 2 by default, so should be set to 0 if no axis labels are present.
rowCurveLabel (optional)	integer	Index of the row containing curve labels. This is row 1 by default, so should be set to 0 if no curve labels are present.
separator (optional)	integer	Separator. Can be Read.CSV_COMMA, Read.CSV_SPACE or Read.CSV_TAB
xColIndex (optional)	integer	Index of the column containing X-values. This is column 1 by default.
xInterval (optional)	real	User defined X-interval between points, to use together with xStartVal
xStartVal (optional)	real	Instead of taking X-values from the CSV file, this allows the user to define a value for the start of the X-axis.

Returns

No return value

Example

To read an XYYY CSV file with X-values in column 3, named "example.csv" starting at curve 5.

```
var options = new Object();
options.csvType = Read.CSV_XYYY;
options.xColIndex = 3;
options.outputOpt = "#5";
Read.CSV("example.csv", options);
```

Cur(Filename*[String]*, Options (optional)*[object]*) [static]

Description

Reads a Curve file into T/HIS.

Arguments

- **Filename** (String)

Name of Curve file to read

- **Options (optional)** (object)

Options which give you greater control of reading a Curve file:

Object has the following properties:

Name	Type	Description
outputOpt (optional)	integer/string	Allows you to control which curve ID number the file will begin to read at. This can either be a whole number greater than 0 or specified as a string like "#3" for curve 3. There are also the special string characters "%", standing for highest free curve as well as "#", standing for the lowest free curve.

Returns

No return value

Example

To read a Curve file named "example.cur"

```
Read.Cur("example.cur");
```

To read a Curve file named "example.cur" at the curve ID 15

```
var options = new Object();
options.outputOpt = 15;
Read.Cur("example.cur",options);
```

DIAdem(Filename*[String]*, X-axis channel*[integer]*, Options (optional)*[object]*) [static]

Description

Reads a DIAdem file into T/HIS.

Arguments

- **Filename** (String)

Name of DIAdem header file to read.

- **X-axis channel** (integer)

Index of the channel to use as X-axis values. If this is 0 then the X-values can be generated from a start value and an interval in the following two arguments.

- **Options (optional)** (object)

Options which give you greater control of reading the Diadem file:

Object has the following properties:

Name	Type	Description
filter (optional)	String	String to filter channel names/comments. Only channels whose names/comments contain the filter string will be read.
outputOpt (optional)	integer/string	Allows you to control which curve ID number the file will begin to read at. This can either be a whole number greater than 0 or specified as a string like "#3" for curve 3. There are also the special string characters "%", standing for highest free curve as well as "#", standing for the lowest free curve. By default, this will be set to the highest free curve.
showChannelName (optional)	real	Option to select what is written to the curve tag. Can be Read.DIADEM_COMMENT or Read.DIADEM_NAME
xAxisInterval (optional)	real	User defined interval between points on the X-axis, to use together with the previous argument.
xAxisStartVal (optional)	real	Instead of taking X-values from a DIAdem channel, this allows the user to define a value for the start of the X-axis.

Returns

No return value

Example

To read DIAdem channels from "EXAMPLE.DAT", with channel comments filtered by "EXAMPLE" and X-values taken from channel 1, starting at curve 10 onwards sequentially:

```
var options = new Object();
options.filter = "EXAMPLE";
options.outputOpt = 10;
Read.DIAdem("./files/DIAdem/EXAMPLE.DAT",1,options);
```

Equation(Formula*[String]*, Options (optional)*[object]*) [static]

Description

Create a curve from a user-defined equation.

Arguments

- **Formula** (String)

Equation string.

- **Options (optional)** (object)

Options which give you greater control of reading the Diadem file:

Object has the following properties:

Name	Type	Description
curveID (optional)	integer	Allows you to control which curve ID the file will begin to read into. By default, this will be set to the highest free curve by default if the option is not selected.
xEndVal (optional)	real	Right endpoint of the x range. Default 1.0.
xInterval (optional)	real	Interval between points. Default 0.01.
xStartVal (optional)	real	Left endpoint of the x range. Default 0.0.
xValOpt (optional)	integer	Option to select how x values are determined for the given equation. Read.EQUATION_X_VALS, requires the xStartVal, xEndVal and xInterval options to also be set however these default to 0, 1 and 0.01 respectively. Read.EQUATION_CURVE_VARS will calculate it's x values from the curve variables that are used in the equation. This is the default if curve variables are used and no xValOpt has been provided. Read.EQUATION_X_OR_CURVE will use Rea.EQUATION_X_VALS if there are no curve variables in the equation otherwise Read.EQUATION_CURVE_VARS if there are. -ID to take x values from Curve #ID (e.g. -5 for curve ID 5)

Returns

No return value

Example

To plot the line y = x^2+2 for x in [-1,1] and an interval between points of 0.02 and set this curve as ID 3:

```
var options = new Object();
options.xStartVal = -1;
options.xEndVal = 1;
options.xInterval = 0.02;
options.xValOpt = Read.EQUATION_X_VALS;
options.curveID = 3;
Read.Equation("x^2+2",options);
```

ISO(Filename*[String]*, Options (optional)*[object]*) [static]

Description

Reads an ISO file into T/HIS.

Arguments

- **Filename** (String)

Name of ISO file to read

- **Options (optional)** (object)

Options which give you greater control of reading an ISO file:

Object has the following properties:

Name	Type	Description
fileFormat (optional)	integer	Format of ISO file. Can be Read.MULTIPLE_CHANNELS, Read.ISO_SINGLE_CHANNEL
labelType (optional)	integer	Label type to use. Can be Read.ISO_CHANNEL_LABEL, Read.ISO_CHANNEL_CODE
outputOpt (optional)	integer/string	Allows you to control which curve ID the file will begin to read to. This can either be a whole number greater than 0 or specified as a string like "#3" for curve 3. There are also the special string characters "%", standing for highest free curve as well as "#", standing for the lowest free curve. By default, this will be set to the highest free curve.

Returns

No return value

Example

To read a single channel ISO file named "example.001" to curve ID 7

```
var options = new Object();
options.fileFormat = Read.ISO_SINGLE_CHANNEL;
options.outputOpt = 7;
Read.ISO("example.001", options);
```

Key(Filename*[String]*, Options (optional)*[object]*) [static]

Description

Reads a Keyword file into T/HIS.

Arguments

- **Filename** (String)

Name of Keyword file to read

- **Options (optional)** (object)

Options which give you greater control of reading a Keyword file:

Object has the following properties:

Name	Type	Description
outputOpt (optional)	integer/string	Allows you to control which curve ID number the file will begin to read at. This can either be a whole number greater than 0 or specified as a string like "#3" for curve 3. There are also the special string characters "%", standing for highest free curve as well as "#", standing for the lowest free curve.

Returns

No return value

Example

To read a Keyword file named "example.key"

```
Read.Key("example.key");
```

To read a Keyword file named "example.key" starting at curve 5 and sequentially incrementing if the file contains multiple curves.

```
var options = new Object();
options.outputOpt = 5;
Read.Key("example.key",options);
```

LSPP(Filename*[String]*, Options (optional)*[object]*) [static]

Description

Reads an LS-PREPOST file into T/HIS.

Arguments

- **Filename** (String)

Name of LS-PREPOST file to read

- **Options (optional)** (object)

Options which give you greater control of reading a Keyword file:

Object has the following properties:

Name	Type	Description
fileFormat (optional)	integer	LSPP file format. Can be Read.LSPP_CURVE_FILE or Read.LSPP_XY_PAIRS
outputOpt (optional)	integer/string	Allows you to control which curve ID number the file will begin to read at. This can either be a whole number greater than 0 or specified as a string like "#3" for curve 3. There are also the special string characters "%", standing for highest free curve as well as "#", standing for the lowest free curve. By default, this will be set to the highest free curve.

Returns

No return value

Example

To read an LS-PREPOST XY pairs file named "example.xy"

```
var options = new Object();
options.fileFormat = Read.LSPP_XY_PAIRS;
Read.LSPP("example.xy", options);
```

Ssh class

The Ssh class allows you to connect to a remote computer using ssh, scp and sftp commands. More...

The T/HIS JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Member functions

- AuthenticateWithPassword(password*[string]*)
- AuthenticateWithPublicKey(passphrase (optional)*[string]*)
- Execute(data*[object]*)
- Get(remote*[string]*, local*[string]*)
- Put(remote*[string]*, local*[string]*)
- SftpGet(remote*[string]*, local*[string]*)
- SftpList(remote*[string]*)
- SftpMkdir(remote*[string]*, mode*[constant]*)
- SftpPut(remote*[string]*, local*[string]*)
- SftpRmdir(remote*[string]*)

Ssh constants

Constants for file bits

Name	Description
Ssh.SETGROUP_BIT	Set group bit
Ssh.SETUID_BIT	Set uid bit
Ssh.STICKY_BIT	sticky bit

Constants for file types

Name	Description
Ssh.DIRECTORY	Directory
Ssh.FILE	Regular file
Ssh.SOCKET	Socket
Ssh.SYMBOLIC_LINK	Symbolic link

Constants for permissions

Name	Description
Ssh.GROUP_EXECUTE	Group has execute permission
Ssh.GROUP_READ	Group has read permission
Ssh.GROUP_WRITE	Group has write permission
Ssh.OTHER_EXECUTE	Others have execute permission
Ssh.OTHER_READ	Others have read permission

Ssh.OTHER_WRITE	Others have write permission
Ssh.OWNER_EXECUTE	Owner has execute permission
Ssh.OWNER_READ	Owner has read permission
Ssh.OWNER_WRITE	Owner has write permission

Detailed Description

The Ssh class gives you simple functions to do secure connections to a remote computer using ssh. Oasys LS-DYNA Environment software is built with the OpenSSH library to support the ssh, scp and sftp protocols. The basic workflow is to create a connection using the Ssh constructor, authenticate the connection either by using a password and AuthenticateWithPassword or with a public key and AuthenticateWithPublicKey then the method Execute can be used to execute commands on the remote machine, the methods Get and Put can be used to copy files to and from the remote machine using scp, and the commands SftpGet, SftpList, SftpMkdir, SftpPut and SftpRmdir can be used to perform secure file transfer commands.
ssh uses a public and private key pair to do communication. The software uses RSA for the private and public keys and stores them in the files id_rsa and id_rsa.pub in the .oasys_ssh directory of your home directory (C:\Users\your.name\.oasys_ssh on Windows by default). A key length of 2048 bits is recommended. You keep your private key secure in your .oasys_ssh directory but the public key can be copied to the authorized_keys file on remote machines so that authentication can be done etc. The software also maintains fingerprints for the machines you connect to to ensure that you are connecting to the machine that you think you are. The first time you connect to a machine you are asked to confirm the remote machine is correct and the software stores the fingerprint for it in the known_hosts file in your .oasys_ssh directory. For second and subsequent connections the software checks the fingerprint of the remote machine against the one it has stored and will only connect if it matches.
When creating a new ssh connection to a remote machine and transferring files a small 'buffer' is required to transfer the data. The size of this buffer can be controlled using the Options.ssh_buffer_size property **before** the Ssh object is created.

Constructor

new Ssh(hostname*[string]*, username*[string]*)

Description

Create a new Ssh object for secure communication to a remote computer.

Arguments

- **hostname** (string)

The hostname of the machine that you want to connect to

- **username** (string)

The username on the machine that you want to connect to

Returns

Ssh object

Return type

Ssh

Example

To create a connection to machine "example" as user "username"

```
var s = new Ssh("example", "username");
```

Details of functions

AuthenticateWithPassword(password*[string]*)

Description

Authenticate the connection using password.

Arguments

- **password** (string)

The password for the username on the remote machine

Returns

no return value

Example

To prompt the user for a password and authenticate using it in SSH connection s:

```
var password = Window.GetPassword("Enter Password to connect", "Password");
s.AuthenticateWithPassword(password);
```

AuthenticateWithPublicKey(passphrase (optional)*[string]*)

Description

Authenticate the connection using your public key. Your public key from the file .oasys_ssh/id_rsa.pub must be in the file .oasys_ssh/authorized_keys on the remote machine.

Arguments

- **passphrase (optional)** (string)

The passphrase for authentication on the remote machine if required

Returns

no return value

Example

Authenticate using your public key in SSH connection s:

```
s.AuthenticateWithPublicKey();
```

Execute(data*[object]*)

Description

Execute a command in the ssh session and get the standard output and error streams.

Arguments

- **data** (object)

Execute data

Object has the following properties:

Name	Type	Description
arguments (optional)	Array of strings	The arguments to pass to the command

command	string	The command you want to run

Returns

Object with the following properties:

Name	Type	Description
status	integer	The exit code from the command
stderr	string	The standard error output from the command
stdout	string	The standard output from the command

Return type

object

Example

To run command "example.bat" with arguments "foo" and "bar" in SSH connection s:

```
var output = s.Execute( { command: 'example.bat', arguments: [ 'foo', 'bar' ]}
);
var text   = output.stdout;
var errors = output.stderr;
var ecode  = output.status;
```

Get(remote*[string]*, local*[string]*)

Description

Gets a file from the ssh connection using scp.

Arguments

- **remote** (string)

The path of the remote file to get

- **local** (string)

The path of the local file to write

Returns

no return value

Example

To get the remote file "/path/to/file.txt", creating local file "C:\path\to\file.txt" in SSH connection s:

```
s.Get("/path/to/file.txt", "C:\\path\\to\\file.txt");
```

Put(remote*[string]*, local*[string]*)

Description

Puts a file on the remote ssh connection using scp.

Arguments

- **remote** (string)

The path of the remote file to put

- **local** (string)

The path of the local file to read

Returns

no return value

Example

To put the local file "C:\path\to\file.txt" to remote file "/path/to/file.txt" in SSH connection s:

```
s.Put("/path/to/file.txt", "C:\\path\\to\\file.txt");
```

SftpGet(remote*[string]*, local*[string]*)

Description

Gets a file from the ssh connection using sftp.

Arguments

- **remote** (string)

The path of the remote file to get

- **local** (string)

The path of the local file to write

Returns

no return value

Example

To get the remote file "file.txt", creating local file "C:\path\to\file.txt" in SSH connection s using sftp:

```
s.SftpGet("file.txt", "C:\\path\\to\\file.txt");
```

SftpList(remote*[string]*)

Description

Gets a listing from the ssh connection using sftp.

Arguments

- **remote** (string)

The remote path to get the listing from

Returns

Array of objects. Each object contains the following information for a file/directory:

Name	Type	Description
atime	integer	Access time for the file (seconds since epoch)
gid	integer	The group ID
info	constant	Bitwise information for the file/directory. See the permissions, file types and file bits constants
mtime	integer	Modification time for the file (seconds since epoch)
name	string	The name of the file/directory
size	integer	The size of the file
uid	integer	The user ID

Return type

object

Example

To get listing from the the remote path "temp" in SSH connection s using sftp:

```
var listing = s.SftpList("temp");
for (l=0; l<listing.length; l++)
{
    Message(listing[l].name + ":" + listing[l].size;
}
```

SftpMkdir(remote*[string]*, mode*[constant]*)

Description

Creates a directory in the remote ssh connection using sftp.

Arguments

- **remote** (string)

The remote directory to create

- **mode** (constant)

The mode/permissions for the directory. See the permissions constants for details. Note that the user's file-creation mask (umask) value will also be taken into account when creating the directory.

Returns

true if successful, false if not

Return type

Boolean

Example

To create the remote path "temp" with, create, write and execute permissions for only the owner, in SSH connection s using sftp:

```
var success = s.SftpMkdir("temp", Ssh.OWNER_READ|Ssh.OWNER_WRITE|Ssh.OWNER_
EXECUTE);
```

SftpPut(remote*[string]*, local*[string]*)

Description

Puts a file on the remote ssh connection using sftp.

Arguments

- **remote** (string)

The path of the remote file to put

- **local** (string)

The path of the local file to read

Returns

no return value

Example

To put the local file "C:\path\to\file.txt" to remote file "file.txt" in SSH connection s:

```
s.SftpPut("file.txt", "C:\\path\\to\\file.txt");
```

SftpRmdir(remote*[string]*)

Description

Deletes a directory in the remote ssh connection using sftp. If this fails it is probably because the directory is not empty.

Arguments

- **remote** (string)

The remote directory to delete

Returns

true if successful, false if not

Return type

Boolean

Example

To delete the remote path "temp" in SSH connection s using sftp:

```
var success = s.SftpRmdir("temp");
```

Symbol class

The Symbol class contains constants relating to curve symbols. More...

The T/HIS JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Symbol constants

Name	Description
Symbol.CIRCLE	Circle symbol
Symbol.CROSS	Cross symbol
Symbol.DIAMOND	Diamond symbol
Symbol.DOT	Dot symbol
Symbol.HOURGLASS	Hourglass symbol
Symbol.NONE	No symbol
Symbol.SQUARE	Square symbol
Symbol.STAR	Star symbol
Symbol.TRIANGLE	Triangle symbol

Detailed Description

The Symbol class is used to define the symbol style used by curves:

```
p.symbol = Symbol.TRIANGLE;
```

Units class

The Units class contains constants relating to curve units. More...

The T/HIS JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- GetDisplayUnits()
- SetDisplayUnits(unit system*[String]*)
- USER(mass*[real]*, time*[real]*, length*[real]*, angle*[real]*, temperature*[real]*, current (optional)*[real]*)

Units constants

Name	Description
Units.ACCELERATION	Acceleration units
Units.AREA	Area units
Units.CHARGE	Charge units
Units.CONDUCTIVITY	Conductivity units
Units.CURRENT	Current units
Units.DENSITY	Density units
Units.DISPLACEMENT	Displacement units
Units.ELECTRIC_FIELD_VECTOR	Electric Field Vector units
Units.ENERGY	Energy units
Units.ENERGY_DENSITY	Energy Density units
Units.FLOW_RATE	Flow Rate units
Units.FLUX	Thermal Flux units
Units.FORCE	Force units
Units.FORCE_WIDTH	Force per unit width units
Units.FREQUENCY	Frequency units
Units.HEAT_TRANSFER_COEFF	Heat Transfer coefficient units
Units.INDUCTANCE	Inductance units
Units.INERTIA	Inertia units
Units.LENGTH	Length units
Units.MAGNETIC_FLUX_VECTOR	Magnetic Flux Vector units
Units.MASS	MAss units
Units.MASS_FLOW	Mass Flow rate units
Units.MOMENT	Moment units
Units.MOMENTUM	Momentum units

Units.MOMENT_WIDTH	Moment per unit width units
Units.NONE	No units
Units.POWER	Power units
Units.PRESSURE	Pressure units
Units.Q_CRITERION	Q Criterion units
Units.RESISTANCE	Resistance units
Units.ROTATION	Rotation units
Units.ROTATIONAL_ACCELERATION	Rotational Acceleration units
Units.ROTATIONAL_VELOCITY	Rotational Velocity units
Units.STRAIN	Strain units
Units.STRESS	Stress units
Units.TEMPERATURE	Temperature units
Units.THERMAL_DIFFUSIVITY	Thermal Diffusivity units
Units.TIME	Time units
Units.UNKNOWN	Unknown units
Units.VECTOR_POTENTIAL	Vector Potential units
Units.VELOCITY	Velocity units
Units.VISCOSITY	Viscosity units
Units.VOLTAGE	Voltage units
Units.VOLUME	Volume units
Units.VORTICITY	Vorticity units
Units.WORK	Work units

Detailed Description

The Units class is used to define the units for each axis of a curve:

```
p.x_axis_units = Units.LENGTH
```

Details of functions

GetDisplayUnits() [static]

Description

Returns the Display units

Arguments

No arguments

Returns

String indicating the display unit system

Return type

String

Example

To get the display units

```
var units = Units.GetDisplayUnits();
```

SetDisplayUnits(unit system*[String]*) [static]

Description

Sets the display units to the units provided by the user

Arguments

- **unit system** (String)

The unit system you want to set the display units to

Returns

TRUE if the Display units are set successfully else FALSE

Return type

Boolean

Example

To set the display units to "U2"

```
Units.SetDisplayUnits("U2");
```

USER(mass*[real]*, time*[real]*, length*[real]*, angle*[real]*, temperature*[real]*, current (optional)*[real]*) [static]

Description

Setup a user defined UNIT

Arguments

- **mass** (real)

Power for mass dimensions.

- **time** (real)

Power for time dimensions.

- **length** (real)

Power for length dimensions.

- **angle** (real)

Power for angle dimensions.

- **temperature** (real)

Power for temperature dimensions.

- **current (optional)** (real)

Power for current dimensions.

Returns

integer

Return type

Number

Example

To set the y-axis unit of curve l to (m/s)^2:

```
l.y_unit = Units.USER(0.0,2.0,-2.0,0.0,0.0,0.0);
```

UnitSystem class

The UnitSystem class contains constants relating to curve unit systems. More...

The T/HIS JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

UnitSystem constants

Name	Description
UnitSystem.U1	U1 unit system (m, kg, s)
UnitSystem.U2	U2 unit system (mm, t, s)
UnitSystem.U3	U3 unit system (mm, kg, ms)
UnitSystem.U4	U4 unit system (mm, g, ms)
UnitSystem.U5	U5 unit system (ft, slug, s)
UnitSystem.U6	U6 unit system (m, t, s)

Detailed Description

The UnitSystem class is used to define the Unit System for a curve:

```
p.UnitSystem = UnitSystem.U1
```

Utils class

The Utils class contains various useful utility functions. More...

The T/HIS JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Ascii85Decode(encoded*[string]*)
- Ascii85Encode(data*[ArrayBuffer]*, length (optional)*[integer]*)
- Build()
- CallPromiseHandlers()
- CheckinLicense(feature*[string]*)
- CheckoutLicense(feature*[string]*)
- GarbageCollect()
- HTMLBrowser()
- HiResTimer()
- PdfReader()
- SHA256(filename*[string]*)
- SHA512(filename*[string]*)
- Terminate()
- TimerResolution()
- UUID()
- Version()

Detailed Description

The Utils class is used to provide various useful functions.

Details of functions

Ascii85Decode(encoded*[string]*) [static]

Description

Decodes an ASCII85 encoded string. See Utils.Ascii85Encode() for details on the method.

Arguments

- **encoded** (string)

An ASCII85 encoded string

Returns

ArrayBuffer object

Return type

ArrayBuffer

Example

To decode an ASCII85 encoded string:

```
var decoded = Utils.Ascii85Decode(encoded);
```

Ascii85Encode(data*[ArrayBuffer]*, length (optional)*[integer]*) [static]

Description

Encodes an ASCII85 encoded string. This enables binary data to be represented by ASCII characters using five ASCII characters to represent four bytes of binary data (making the encoded size 1/4 larger than the original). By doing this binary data can be stored in JavaScript strings. Note that the method used by THIS to encode and decode strings differs from the standard ASCII85 encoding as that uses the ASCII characters ", ’ and \ which cannot be used in JavaScript strings as they have special meanings. The method in THIS uses
0-84 are !-u (ASCII codes 33-117) (i.e. 33 is added to it) with the following exceptions
v is used instead of " (ASCII code 118 instead of 34)
w is used instead of ’ (ASCII code 119 instead of 39)
x is used instead of \ (ASCII code 120 instead of 92)
If all five digits are 0 they are represented by a single character z instead of !!!!!

Arguments

- **data** (ArrayBuffer)

ArrayBuffer containing the data

- **length (optional)** (integer)

Length of data in array buffer to encode. If omitted the whole array buffer will be encoded

Returns

string

Return type

String

Example

To encode ArrayBuffer data:

```
var encoded = Utils.Ascii85Encode(data);
```

Build() [static]

Description

Returns the build number

Arguments

No arguments

Returns

integer

Return type

Number

Example

To get the current build number

```
var build = Utils.Build();
```

CallPromiseHandlers() [static]

Description

Manually call any promise handlers/callbacks in the job queue

Arguments

No arguments

Returns

no return value

Example

To run any queued promise handlers/callbacks:

```
Utils.CallPromiseHandlers();
```

CheckinLicense(feature*[string]*) [static]

Description

Checks a license for a feature back in

Arguments

- **feature** (string)

feature to check license back in for

Returns

no return value

Example

To check in a license for "EXAMPLE":

```
Utils.CheckinLicense("EXAMPLE");
```

CheckoutLicense(feature*[string]*) [static]

Description

Checks out a license for a feature

Arguments

- **feature** (string)

feature to check license for

Returns

true if license available, false if not

Return type

Boolean

Example

To checkout a license for "EXAMPLE":

```
var got = Utils.CheckoutLicense("EXAMPLE");
if (got == false) Exit();
```

GarbageCollect() [static]

Description

Forces garbage collection to be done. This should not normally need to be called but in exceptional circumstances it can be called to ensure that garbage collection is done to return memory.

Arguments

No arguments

Returns

no return value

Example

To force garbage collection to be done:

```
Utils.GarbageCollect();
```

HTMLBrowser() [static]

Description

Returns the path to the default HTML browser

Arguments

No arguments

Returns

string of the path

Return type

String

Example

To get path to the default HTML browser

```
var path = Utils.HTMLBrowser();
```

HiResTimer() [static]

Description

A high resolution timer that can be used to time how long things take. The first time this is called the timer will start and return 0. Subsequent calls will return the time in nanoseconds since the first call. Note that the timer will almost certainly not have 1 nanosecond precision but, depending on the platform, should should have a resolution of at least 1 microsecond. The resolution can be found by using Utils.TimerResolution()

Arguments

No arguments

Returns

number

Return type

number

Example

To time how long something takes to nanosecond precision:

```
var start = Utils.HiResTimer();
do something that takes some time...
var end = Utils.HiResTimer();
Message("it took " + (end-start) + "nanoseconds");
```

PdfReader() [static]

Description

Returns the path to the executable of the default pdf reader

Arguments

No arguments

Returns

string of the path

Return type

String

Example

To get path to the default pdf reader

```
var path = Utils.PdfReader();
```

SHA256(filename*[string]*) [static]

Description

Create a SHA-256 hash for a file

Arguments

- **filename** (string)

File to calculate the hash for

Returns

string

Return type

String

Example

To generate the hash for file "example.txt":

```
var sha256 = Utils.SHA256("example.txt");
```

SHA512(filename*[string]*) [static]

Description

Create a SHA-512 hash for a file

Arguments

- **filename** (string)

File to calculate the hash for

Returns

string

Return type

String

Example

To generate the hash for file "example.txt":

```
var sha512 = Utils.SHA512("example.txt");
```

Terminate() [static]

Description

Terminates the program. Note that this **terminates the entire program**, not the script. To terminate the script use Exit() instead

Arguments

No arguments

Returns

No return value

Example

To terminate the program:

```
Utils.Terminate();
```

TimerResolution() [static]

Description

Returns the resolution (precision) of the Utils.HiResTimer() timer in nanoseconds

Arguments

No arguments

Returns

number

Return type

number

Example

To find the resolution of the timer in nanoseconds:

```
var resolution = Utils.TimerResolution();
```

UUID() [static]

Description

Create an UUID (Universally Unique Identifier)

Arguments

No arguments

Returns

string

Return type

String

Example

To generate an UUID:

```
var uuid = Utils.UUID();
```

Version() [static]

Description

Returns the version number

Arguments

No arguments

Returns

real

Return type

Number

Example

To get the current version number

```
var version = Utils.Version();
```

Widget class

The Widget class allows you to create components for a graphical user interface. More...

The T/HIS JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- CtrlPressed()
- PixelsPerUnit()
- ShiftPressed()
- StringLength(text*[string]*, monospace (optional)*[boolean]*, fontSize (optional)*[integer]*)

Member functions

- AddWidgetItem(item*[WidgetItem]*, position (optional)*[integer]*)
- AddWidgetItem(item*[WidgetItem]*, relationship*[constant]*, relitem*[WidgetItem]*)
- Circle(colour*[constant]*, fill*[boolean]*, xc*[integer]*, yc*[integer]*, radius*[integer]*)
- Clear()
- ClearSelection()
- Cross(colour (optional)*[constant]*)
- Delete()
- DirectoryIcon(line_colour*[constant]*, fill_colour*[constant]*)
- DumpImageString(filename*[string]*, format (optional)*[constant]*)
- Hide()
- ItemAt(index*[integer]*)
- Line(colour*[constant]*, x1*[integer]*, y1*[integer]*, x2*[integer]*, y2*[integer]*)
- MoveWidgetItem(item*[WidgetItem]*, relationship*[constant]*, relitem*[WidgetItem or null]*)
- Polygon(colour*[constant]*, fill*[boolean]*, points*[Array of numbers]*)
- Polygon(colour*[constant]*, fill*[boolean]*, x1*[integer]*, y1*[integer]*, x2*[integer]*, y2*[integer]*, ... xn*[integer]*, ... yn*[integer]*) [deprecated]
- ReadImageFile(filename*[string]*, justify (optional)*[constant]*, transparent (optional)*[colour value (integer)]*, tolerance (optional)*[integer]*)
- ReadImageString(string*[string]*, justify (optional)*[constant]*, transparent (optional)*[colour value (integer)]*, tolerance (optional)*[integer]*)
- Rectangle(colour*[constant]*, fill*[boolean]*, x1*[integer]*, y1*[integer]*, x2*[integer]*, y2*[integer]*)
- RemoveAllWidgetItems()
- RemoveWidgetItem(item*[WidgetItem]*)
- Scroll(scroll*[constant or WidgetItem object]*)
- Show()
- Static()
- Tick(colour (optional)*[constant]*)
- TotalItems()
- WidgetItems()

Widget constants

Name	Description
Widget.BUTTON	Button widget
Widget.CHECKBOX	Checkbox widget
Widget.COMBOBOX	Combobox widget
Widget.LABEL	Label widget
Widget.LISTBOX	Listbox widget

Widget.RADIOBUTTON	Radiobutton widget
Widget.SLIDER	Slider widget
Widget.TEXTBOX	Text input widget
Widget.TREE	Tree widget

Constants for Colour

Name	**Description**
Widget.BLACK	Colour black
Widget.BLUE	Colour blue
Widget.COLOUR_ CONTRAST	A contrasting colour in the 3 user interface themes (Green, Purple, and Blue in the Dark, Light, and Classic themes respectively). Blue in the legacy theme.
Widget.COLOUR_ CONTRAST_2	Another contrasting colour in the 3 user interface themes (Yellow, Red, and Red in the Dark, Light, and Classic themes respectively). Red in the legacy theme.
Widget.COLOUR_ INVERSE	Inverse colour in the 3 user interface themes (Black or white depending on theme). Black in the legacy theme.
Widget.COLOUR_LABEL	Label text colour in the 3 user interface themes (Black or white depending on theme). Black in the legacy theme.
Widget.COLOUR_ LATENT	Latent colour in the 3 user interface themes (Different shade of Cyan in every theme). Light Cyan in the legacy theme.
Widget.COLOUR_ NEUTRAL	Neutral colour in the 3 user interface themes (Different shade of grey in every theme). Light grey in the legacy theme.
Widget.COLOUR_SAFE	Safe colour in the 3 user interface themes (Different shade of green in every theme). Dark green in the legacy theme.
Widget.COLOUR_TITLE	Title colour in the 3 user interface themes (Different shade of grey in every theme). Dark blue in the legacy theme.
Widget.COLOUR_ WARNING	Warning colour in the 3 user interface themes (Different shade of red in every theme). Dark red in the legacy theme.
Widget.CYAN	Colour cyan
Widget.DARKBLUE	Colour dark blue
Widget.DARKGREEN	Colour dark green
Widget.DARKGREY	Colour dark grey
Widget.DARKGREY_ NEUTRAL	Only valid in the function ’Line’. Used to keep the 3D effect in the legacy theme and not in the other themes. Neutral colour in the 3 user interface themes (Different shade of grey in every theme). Dark grey in the legacy theme
Widget.DARKRED	Colour dark red
Widget.DEFAULT	Default colour for widgets
Widget.GREEN	Colour green
Widget.GREY	Colour grey
Widget.LIGHTGREY	Colour light grey
Widget.LIGHTGREY_ NEUTRAL	Only valid in the function ’Line’. Used to keep the 3D effect in the legacy theme and not in the other themes. Neutral colour in the 3 user interface themes (Different shade of grey in every theme). Light grey in the legacy theme
Widget.MAGENTA	Colour magenta
Widget.ORANGE	Colour orange

Widget.RED	Colour red
Widget.WHITE	Colour white
Widget.YELLOW	Colour yellow

Constants for Image RGB format

Name	Description
Widget.RGB24	24 bits for RGB data in widget images
Widget.RGB8	8 bits for RGB data in widget images

Constants for Justification

Name	Description
Widget.BOTTOM	Bottom justification
Widget.CENTRE	Centre (horizontal) justification
Widget.LEFT	Left justification
Widget.MIDDLE	Middle (vertical) justification
Widget.RIGHT	Right justification
Widget.SCALE	Image will be scaled to fit widget
Widget.TOP	Top justification

Constants for Orientation

Name	Description
Widget.HORIZONTAL	Horizontal orientation (for sliders)
Widget.VERTICAL	Vertical orientation (for sliders)

Constants for Selection

Name	Description
Widget.SELECT_ENHANCED	Multiple WidgetItems in a ListBox or tree Widget can be selected. When the user selects a WidgetItem the selection is cleared and the new WidgetItem selected. However, if the user presses the Ctrl key when clicking on a WidgetItem, the clicked WidgetItem gets toggled and all other WidgetItems are left untouched. If the user presses the Shift key while clicking on a WidgetItem, all WidgetItems between the last selected WidgetItem and the clicked WidgetItem are selected or unselected, depending on the state of the clicked WidgetItem.
Widget.SELECT_MULTIPLE	Multiple WidgetItems in a ListBox Widget can be selected. When the user selects a WidgetItem, the selection status of that WidgetItem is toggled and the other WidgetItems are left alone. Not valid for tree widgets. Widget.SELECT_ENHANCED will be used.
Widget.SELECT_NONE	No WidgetItem in a ListBox or tree Widget can be selected
Widget.SELECT_SINGLE	A single WidgetItem in a ListBox or tree Widget can be selected. When the user selects a WidgetItem, any already-selected WidgetItem becomes unselected, and the user cannot unselect the selected WidgetItem by clicking on it.

Constants for Tree relations

Name	Description

Widget.AFTER	Add a WidgetItem after the existing WidgetItem for a tree widget.
Widget.BEFORE	Add a WidgetItem before the existing WidgetItem for a tree widget.
Widget.CHILD	Add a WidgetItem as a child of the existing WidgetItem for a tree widget.

Constants for Tree/listbox scrolling

Name	Description
Widget.SCROLL_BOTTOM	Scroll tree or listbox widget to bottom.
Widget.SCROLL_DOWN	Scroll tree or listbox widget down one.
Widget.SCROLL_PAGE_DOWN	Scroll tree or listbox widget down one page.
Widget.SCROLL_PAGE_UP	Scroll tree or listbox widget up one page.
Widget.SCROLL_TOP	Scroll tree or listbox widget to top.
Widget.SCROLL_UP	Scroll tree or listbox widget up one.

Constants for User interface categories

Name	Description
Widget.CATEGORY_APPLY	Apply buttons
Widget.CATEGORY_BUTTON_BOX	A button box panel that contains other widgets
Widget.CATEGORY_CANCEL	Buttons which cancel the current operation
Widget.CATEGORY_DATA_ ENTRY_HEADER	Header for data entry cells, e.g. PRIMER create panels
Widget.CATEGORY_DISMISS	Buttons to close or dismiss panels
Widget.CATEGORY_ENTITY	Entity types in T/HIS
Widget.CATEGORY_GENERIC	A generic button that isn’t a special category
Widget.CATEGORY_GENERIC_2	An alternative to the generic category that has a complementary colour
Widget.CATEGORY_HELP	Help buttons
Widget.CATEGORY_KEYWORD	A PRIMER keyword button
Widget.CATEGORY_LABEL	A text label
Widget.CATEGORY_LABEL_BOX	Text label with a border
Widget.CATEGORY_LABEL_ POPUP	Text label with a popup that blends into the background
Widget.CATEGORY_MENU_BOX	A menu box
Widget.CATEGORY_MESSAGE	For displaying a temporary warning message
Widget.CATEGORY_OPERATE	Operate buttons in T/HIS
Widget.CATEGORY_POPUP_BOX	A popup box that can contain buttons and plain text
Widget.CATEGORY_SAFE_ ACTION	Buttons (usually green) to indicate a safe action
Widget.CATEGORY_SEL_ALL	Select all

Widget.CATEGORY_TAB	Tab
Widget.CATEGORY_TABLE_HEADER	Table (column) header
Widget.CATEGORY_TABLE_ROW	Table row
Widget.CATEGORY_TEXT_BOX	A text box
Widget.CATEGORY_TICKBOX	A tick box
Widget.CATEGORY_TITLE	Title text
Widget.CATEGORY_TOGGLE	Buttons that can be toggled, e.g. On/Off
Widget.CATEGORY_TOOL	Buttons within the tools area
Widget.CATEGORY_UNDO	Buttons which undo the last operation
Widget.CATEGORY_UNSEL_ALL	Unselect/deslect all
Widget.CATEGORY_UPDATE	Update buttons which update the screen but leave the panel open
Widget.CATEGORY_WARNING_ACTION	Buttons (usually red) to indicate a dangerous action
Widget.NO_CATEGORY	No styling is applied. Widget colour controlled by foreground/background properties and is the same in all themes

Widget properties

Name	Type	Description
active	logical	If widget is active (true) or disabled (false)
arrows	boolean	Whether arrows will be shown for a slider (default is true). Slider Widgets only.
background	constant	Widget background colour. Can be: Widget.BLACK, Widget.WHITE, Widget.RED, Widget.GREEN, Widget.BLUE, Widget.CYAN, Widget.MAGENTA, Widget.YELLOW, Widget.DARKRED, Widget.DARKGREEN, Widget.DARKBLUE, Widget.GREY, Widget.DARKGREY, Widget.LIGHTGREY, Widget.ORANGE, Widget.DEFAULT, Widget.COLOUR_NEUTRAL, Widget.COLOUR_CONTRAST, Widget.COLOUR_CONTRAST_2, Widget.COLOUR_WARNING, Widget.COLOUR_SAFE, Widget.COLOUR_TITLE, Widget.COLOUR_INVERSE, Widget.DARKGREY_NEUTRAL, Widget.LIGHTGREY_NEUTRAL Widget.COLOUR_LATENT, Note, background colours in the Window.THEME_DARK, Window.THEME_LIGHT, and Window.THEME_CLASSIC themes will be determined by the category of the widget not the background colour. To override this behaviour and use this background colour first set the widget category to Widget.NO_CATEGORY.
bottom	integer	Widget bottom coordinate
category	constant	The button category which determines the button's appearance when using the new user interface, see Window.Theme()
currentItem	WidgetItem object	The current WidgetItem for a tree Widget. The current WidgetItem in a tree is shown with a dashed border.
fontSize	integer	Widget font size in points. Currently only supports the following sizes: 6, 7, 8, 10, 12, 14, 18, 24. Can be used only with Widget.LABEL and Widget.BUTTON. Both LATIN1 and UTF-8 encoding is supported on Windows but Linux only supports LATIN1 encoding at the moment.

foreground	constant	Widget foreground colour. Can be: Widget.BLACK, Widget.WHITE, Widget.RED, Widget.GREEN, Widget.BLUE, Widget.CYAN, Widget.MAGENTA, Widget.YELLOW, Widget.DARKRED, Widget.DARKGREEN, Widget.DARKBLUE, Widget.GREY, Widget.DARKGREY, Widget.LIGHTGREY, Widget.ORANGE, Widget.DEFAULT, Widget.COLOUR_NEUTRAL, Widget.COLOUR_CONTRAST, Widget.COLOUR_CONTRAST_2, Widget.COLOUR_WARNING, Widget.COLOUR_SAFE, Widget.COLOUR_TITLE, Widget.COLOUR_LABEL, Widget.COLOUR_INVERSE, Widget.DARKGREY_NEUTRAL, Widget.LIGHTGREY_NEUTRAL Widget.COLOUR_LATENT, Note, foreground colours in the Window.THEME_DARK, Window.THEME_LIGHT, and Window.THEME_CLASSIC themes will be determined by the category of the widget not the foreground colour. To override this behaviour and use this foreground colour first set the widget category to Widget.NO_CATEGORY.
hover	string	Widget hover text
imageHeight (read only)	integer	Height of widget image (pixels)
imageWidth (read only)	integer	Width of widget image (pixels)
justify	constant	Widget justification. Can be: Widget.LEFT, Widget.RIGHT or Widget.CENTRE (default).
left	integer	Widget left coordinate
lineWidth	integer	Width of lines when drawing graphics (initially 1; values 1-100 allowed).
macroTag	string	Tag to use for this widget when recording a macro. If empty then the text property value will be used.
maximum	integer	The maximum value allowed for a slider (default is 100). Slider Widgets only.
minimum	integer	The minimum value allowed for a slider (default is 0). Slider Widgets only.
monospace	boolean	true if the widget uses a monospace font instead of a proportional width font (default). Label and button Widgets only.
onChange	function	Function to call when the text in a TEXTBOX widget, the selection in a COMBOBOX widget or the value of a SLIDER is changed. The Widget object is accessible in the function using the 'this' keyword (see the example below for more details of how to define the function and how to use the 'this' keyword). To unset the function set the property to null. **Note that this function is called when the user actually types something into the textbox, selects an item in the combobox or moves the slider, NOT when the Widget.text or Widget.value property changes**.
onClick	function	Function to call when a BUTTON, CHECKBOX, COMBOBOX, LABEL, RADIOBUTTON or TREE widget is clicked. The Widget object is accessible in the function using the 'this' keyword (see the example below for more details of how to define the function and how to use the 'this' keyword). To unset the function set the property to null. **Note that this function is called when the user actually clicks on the button, NOT when the Widget.pushed property changes**. For the COMBOBOX widget the function is called **before** the list of items is mapped.
onPopup	function	Function to call when a BUTTON, LABEL, TEXTBOX or TREE widget is right clicked to map a popup. The Widget object is accessible in the function using the 'this' keyword. The PopupWindow can then be found by using the popupWindow property of the Widget. The function is called **before** the popup is mapped so you can change the widgets in the popup as required.
onTimer	function	Function to call for a widget when timerDelay ms have elapsed after setting this. Additionally if timerRepeat is set this function will be called repetitively, every timerDelay ms. The Widget object is accessible in the function using the 'this' keyword. To unset the function set the property to null. **Note that as soon as this property is set the timer starts!**

orientation	constant	The orientation of a slider. Can be: Widget.VERTICAL or Widget.HORIZONTAL (default). Slider Widgets only.
popupDirection	constant	How PopupWindow will be mapped relative to this widget. Can be Widget.LEFT, Widget.RIGHT, Widget.TOP or Widget.BOTTOM (default). For tree widgets this will be ignored as the popup is always shown on the WidgetItem that is right clicked.
popupSymbol	logical	TRUE (default) if a symbol will be shown for a PopupWindow.
popupWindow	PopupWindow object	PopupWindow for this Widget. Only available for Button, Label and Textbox Widgets. To remove a PopupWindow from a Widget set to null.
pushed	logical	If widget is pushed (true) or not (false). This only affects Widget.BUTTON with the Widget.toggle property set, and Widget.CHECKBOX widgets.
right	integer	Widget right coordinate
select	constant	Selection method for ListBox and tree Widgets. Can be: Widget.SELECT_NONE, Widget.SELECT_SINGLE or Widget.SELECT_MULTIPLE or Widget.SELECT_ENHANCED (default).
selectedItem	WidgetItem object	WidgetItem that is currently selected for a ComboBox or Radiobutton, Widget. If null no WidgetItem is selected. For a ListBox Widget this property contains the last WidgetItem that was (de)selected. To get a list of all of the selected WidgetItems use WidgetItems() to return all of the WidgetItems and inspect the WidgetItem selected property.
shown (read only)	boolean	true if the widget is visible. To alter the visibility of a widget use the Show() and Hide() methods.
step	integer	The step value of a slider (default is 1). Slider Widgets only.
text	string	Widget text. For a ComboBox Widget this will be the text for the currently selected WidgetItem
textHidden	boolean	true if the widget text is hidden and replaced by asterisks. This may be used to create textboxes to type passwords in. TextBox Widgets only.
timerDelay	integer	Delay in ms before the function set for onTimer will be called. The initial value is 1000 (ms). Also see timerRepeat.
timerRepeat	logical	If the function set for onTimer will be called once (false) or repeatedly (true). The initial value is false. Also see timerDelay.
toggle	logical	If widget can be toggled (true) or not (false). This only affects Widget.BUTTON widgets.
top	integer	Widget top coordinate
type (read only)	integer	Type of the widget. The widget type could be Widget.BUTTON, Widget.CHECKBOX, Widget.COMBOBOX, Widget.LABEL, Widget.LISTBOX, Widget.RADIOBUTTON, Widget.SLIDER, Widget.TEXTBOX or Widget.TREE
value	integer	The current value of a slider (initially will be the minimum value). Slider Widgets only.
window (read only)	Window object	The Window that this widget is defined in
xResolution	integer	X resolution of button when drawing lines, circles, polygons and rectangles (initially 100). X coordinates on the Widget can be from 0 (on the left of the widget) to xResolution (on the right of the widget). Available for Widget.LABEL and Widget.BUTTON Widgets.
yResolution	integer	Y resolution of button when drawing lines, circles, polygons and rectangles (initially 100). Y coordinates on the Widget can be from 0 (on the top of the widget) to yResolution (on the bottom of the widget). Available for Widget.LABEL and Widget.BUTTON Widgets.

Detailed Description

The Widget class allows you to create Widgets (buttons, textboxes etc) in a Window for a graphical user interface. Callback functions can be declared for widgets to give actions when a button is pressed or the text in a textbox is selected etc. The following example displays various widgets in a window. Several callback methods are used. The exit button allows the user to exit the script but the button is only made active if the checkbox widget is ticked. If the button widgets are pressed feedback is given to the user

```
var count = 0;
// Create window
var w = new Window("Test", 0.8, 1.0, 0.5, 0.6);
// Create all of the widgets
var l = new Widget(w, Widget.LABEL,    1, 30, 1, 7, "Text:");
var t = new Widget(w, Widget.TEXTBOX, 31, 80, 1, 7, "Enter text");
var b = new Widget(w, Widget.BUTTON,   1, 30, 8, 14, "Press me");
var b2= new Widget(w, Widget.BUTTON,  31, 61, 8, 14, "Don't press me");
var c = new Widget(w, Widget.CHECKBOX,62, 68, 8, 14);
var l2= new Widget(w, Widget.LABEL,    1, 80, 15, 21, "You haven't pressed the
button yet...");
var e = new Widget(w, Widget.BUTTON,   1, 21, 22, 28, "Exit");
// Allow button widget b2 to toggle
b2.toggle = true;
// The exit button is initially inactive
e.active = false;
// Assign the callback functions
b.onClick  = clicked;
b2.onClick = clicked;
c.onClick  = clicked;
t.onChange = changed;
e.onClick  = confirm_exit;
// Show the window and start event loop
w.Show();
////////////////////////////////////////////////////////////////////////////
function clicked()
{
// If checkbox is clicked then set the state of the exit button
    if (this === c)
    {
        Message("Checkbox clicked");
        e.active = c.pushed;
    }
// If the "Don't press me' button is pressed then change the colour if the
button is pressed in.
    else if (this === b2)
    {
        Message("I said don't press!!!");
        if (b2.pushed) b2.background = Widget.WHITE;
        else           b2.background = Widget.DEFAULT;
    }
// If the "Press me" button is pressed then update the text in the label widget
// with how many times the button has been pressed.
    else
    {
        Message("You pressed...");
        count++;
        l2.text = "Button pressed " + count + " times";
    }
}
////////////////////////////////////////////////////////////////////////////
function changed()
{
// If the user has changed the text in the textbox then give a message in
// the dialogue box
    Message("Text has changed to " + this.text);
}
////////////////////////////////////////////////////////////////////////////
function confirm_exit()
{
// Map confirm box
    var ret = Window.Question("Confirm exit", "Are you sure you want to quit?");
```

```
// If the user has answered yes then exit from the script.
    if (ret == Window.YES) Exit();
}
```

In version 20 a tree widget was added. A simple tree widget example is shown below.

```
Window.Theme(Window.THEME_CURRENT);
let wi, cwi;
// Create a popup window and some widgets
let pw = new PopupWindow();
let pw_l1 = new Widget(pw, Widget.LABEL, 1, 61,  1,  7, "");
let pw_l2 = new Widget(pw, Widget.LABEL, 1, 61,  7, 13, "");
let pw_l3 = new Widget(pw, Widget.LABEL, 1, 61, 13, 19, "");
let pw_l4 = new Widget(pw, Widget.LABEL, 1, 61, 19, 25, "");
let pw_l5 = new Widget(pw, Widget.LABEL, 1, 61, 25, 31, "");
let pw_l6 = new Widget(pw, Widget.LABEL, 1, 61, 31, 37, "");
// Create window
let w = new Window("JavaScript Tree widget test", 0.85, 1.0, 0.75, 1.0);
// Create tree widget
let t = new Widget(w, Widget.TREE, 1, 61, 1, 51, "Suite");
// Add a root node to tree
let env_wi = new WidgetItem(t, "MyItem");
// Add a child to the root node
wi  = new WidgetItem(t, "PRIMER", Widget.CHILD, env_wi);
cwi = new WidgetItem(t, "Prepare", Widget.CHILD, wi);
wi.onMouseOver = wi_onmouseover;
cwi.hover = "Efficient, reliable model setup with support for all of the latest
Ansys LS-DYNA features";
wi  = new WidgetItem(t, "Ansys LS-DYNA", Widget.CHILD, env_wi);
cwi = new WidgetItem(t, "Analyse", Widget.CHILD, wi);
wi.onMouseOver = wi_onmouseover;
// Add a sibling node after Ansys LS-DYNA
wi  = new WidgetItem(t, "REPORTER", Widget.AFTER, wi);
cwi = new WidgetItem(t, "Report", Widget.CHILD, wi);
wi.onMouseOver = wi_onmouseover;
cwi.hover = "Automatic report generation for Ansys LS-DYNA simulations";

// Add a sibling node before REPORTER
wi  = new WidgetItem(t, "T/HIS", Widget.BEFORE, wi);
cwi = new WidgetItem(t, "Process", Widget.CHILD, wi);
wi.onMouseOver = wi_onmouseover;
cwi.hover = "Plot, manipulate and process XY data from Ansys LS-DYNA";

// Alternatively, create WidgetItem without parent and add
let d3plot_wi   = new WidgetItem(null, "D3PLOT");
t.AddWidgetItem(d3plot_wi, Widget.BEFORE, wi);
cwi = new WidgetItem(null, "Visualise");
t.AddWidgetItem(cwi, Widget.CHILD, d3plot_wi);
d3plot_wi.onMouseOver = wi_onmouseover;
cwi.hover = " In-depth 3D visualisation of Ansys LS-DYNA results";
// Expand root node
env_wi.expanded = true;
// Link popup to tree widget
t.popupWindow = pw;
// Assign callbacks
t.onClick = click_tree;
t.onPopup = do_popup;
// Show the widget and start event loop
w.Show();
////////////////////////////////////////////////////////////////////////////
function do_popup()
{
    pw_l1.text = this.currentItem.text;
    if (this.currentItem.selected) pw_l2.text = "Selected";
    else                           pw_l2.text = "Not selected";
    if (this.currentItem.Parent())
        pw_l3.text = "Parent: " + this.currentItem.Parent().text;
    else
        pw_l3.text = "No parent";
    if (this.currentItem.FirstChild())
        pw_l4.text = "First child: " + this.currentItem.FirstChild().text;
    else
```

```
        pw_l4.text = "No children";
    if (this.currentItem.PreviousSibling())
        pw_l5.text = "Previous: " + this.currentItem.PreviousSibling().text;
    else
        pw_l5.text = "No previous";
    if (this.currentItem.NextSibling())
        pw_l6.text = "Next: " + this.currentItem.NextSibling().text;
    else
        pw_l6.text = "No next";
}
////////////////////////////////////////////////////////////////////////////
function click_tree()
{
    Message("Clicked on "+this.currentItem.text+" in tree");
}
////////////////////////////////////////////////////////////////////////////
function wi_onmouseover()
{
    Message("Called onMouseOver for WidgetItem "+this.text+" in tree");
}
```

Graphics (lines, circles, rectangles etc) can be drawn on Widget.LABEL and Widget.BUTTON widgets. If these methods are used the resolution of the widget is 100 units in x and y and the origin is at the top left of the widget. See the documentation below and the WidgetItem and Window classes for more details.

Constructor

new Widget(window*[Window or PopupWindow]*, type*[constant]*, left*[integer]*, right*[integer]*, top*[integer]*, bottom*[integer]*, text (optional)*[string]*)

Description

Create a new Widget object.

Arguments

- **window** (Window or PopupWindow)

Window or PopupWindow that widget will be created in

- **type** (constant)

Widget type. Can be Widget.BUTTON, Widget.CHECKBOX, Widget.COMBOBOX, Widget.LABEL, Widget.LISTBOX, Widget.RADIOBUTTON, Widget.SLIDER, Widget.TEXTBOX or Widget.TREE

- **left** (integer)

left coordinate of widget

- **right** (integer)

right coordinate of widget

- **top** (integer)

top coordinate of widget

- **bottom** (integer)

bottom coordinate of widget

- **text (optional)** (string)

Text to show on widget (optional for LABEL, BUTTON, TEXTBOX and TREE, not required for CHECKBOX, COMBOBOX, LISTBOX, RADIOBUTTON and SLIDER). For a TREE widget the text will be used as a macroTag.

Returns

Widget object

Return type

Widget

Details of functions

AddWidgetItem(item*[WidgetItem]*, position (optional)*[integer]*)

Description

Adds a WidgetItem to a ComboBox ListBox or Radiobutton Widget. Also see Widget.RemoveAllWidgetItems and Widget.RemoveWidgetItem.

Arguments

- **item** (WidgetItem)

WidgetItem to add

- **position (optional)** (integer)

Position on Widget to add the WidgetItem. Any existing WidgetItems will be shifted down as required. If omitted the WidgetItem will be added to the end of the existing ones. **Note that positions start at 0**.

Returns

No return value

Example

To add WidgetItem wi to widget w:

```
w.AddWidgetItem(wi);
```

AddWidgetItem(item*[WidgetItem]*, relationship*[constant]*, relitem*[WidgetItem]*)

Description

Adds a WidgetItem to a Tree Widget. Also see Widget.RemoveAllWidgetItems and Widget.RemoveWidgetItem.

Arguments

- **item** (WidgetItem)

WidgetItem to add

- **relationship** (constant)

What relationship (relative to relitem) to use when adding item to the Widget. Can be:
Widget.BEFORE,
Widget.AFTER or
Widget.CHILD.

- **relitem** (WidgetItem)

Existing WidgetItem to add item relative to. If relationship is Widget.CHILD then relitem can be null and then the WidgetItem will be added to the root node of the tree.

Returns

No return value

Example

To add WidgetItem wi to tree widget w after existing WidgetItem ewi:

```
w.AddWidgetItem(wi, Widget.AFTER, ewi);
```

To add WidgetItem wi to tree widget w as a child of existing WidgetItem ewi:

```
w.AddWidgetItem(wi, Widget.CHILD, ewi);
```

Circle(colour*[constant]*, fill*[boolean]*, xc*[integer]*, yc*[integer]*, radius*[integer]*)

Description

Draws a circle on the widget. Only possible for Widget.LABEL and Widget.BUTTON widgets. The coordinates are local to the Widget, not the Window. See properties xResolution and yResolution for more details. Note that the widget graphics will only be updated when the widget is redrawn. This is to allow the user to do multiple drawing commands on a widget. To force the widget to be redrawn call Show().

Arguments

- **colour** (constant)

Colour of circle. See foreground for colours.

- **fill** (boolean)

If circle should be filled or not.

- **xc** (integer)

x coordinate of centre of circle.

- **yc** (integer)

y coordinate of centre of circle.

- **radius** (integer)

radius of circle.

Returns

no return value

Example

To draw a red filled circle, radius 25, at (50, 50) on widget w:

```
w.Circle(Widget.RED, true, 50, 50, 25);
```

Clear()

Description

Clears any graphics on the widget. Only possible for Widget.LABEL and Widget.BUTTON widgets. Note that the widget graphics will only be updated when the widget is redrawn. This is to allow the user to do multiple drawing commands on a widget. To force the widget to be redrawn call Show().

Arguments

No arguments

Returns

no return value

Example

To clear any graphics for widget w:

```
w.Clear();
```

ClearSelection()

Description

Clears selection of any WidgetItems on the widget. Only possible for Widget.COMBOBOX, Widget.LISTBOX, Widget.RADIOBUTTON and Widget.TREE widgets.

Arguments

No arguments

Returns

no return value

Example

To clear selection of any WidgetItems for widget w:

```
w.ClearSelection();
```

Cross(colour (optional)*[constant]*)

Description

Draws a cross symbol on the widget. Only possible for Widget.LABEL and Widget.BUTTON widgets.

Arguments

- **colour (optional)** (constant)

Colour of cross symbol. See foreground for colours. If omitted, current foreground colour is used.

Returns

no return value

Example

To draw a red cross symbol on widget w:

```
w.Cross(Widget.RED);
```

CtrlPressed() [static]

Description

Check to see if the Ctrl key is pressed

Arguments

No arguments

Returns

true/false

Return type

Boolean

Example

To test if someone has the Ctrl key pressed:

```
if (Widget.CtrlPressed()) { ... }
```

Delete()

Description

Deletes the widget from T/HIS (removing it from the window it is defined in) and returns any memory/resources used for the widget. This function should not normally need to be called. However, sometimes a script may want to recreate widgets in a window many times and unless the old widgets are deleted T/HIS will reach the maximum number of widgets for a window (Options.max_widgets). To avoid this problem this method can be used to force T/HIS to delete and return the resources for a widget. **Do not use the Widget object after calling this method**.

Arguments

No arguments

Returns

no return value

Example

To delete widget w:

```
w.Delete();
```

DirectoryIcon(line_colour*[constant]*, fill_colour*[constant]*)

Description

Draws a directory icon on the widget. Only possible for Widget.BUTTON widgets.

Arguments

- **line_colour** (constant)

Colour of lines of folder (only used in the old UI - in the new UI it will be ignored, a standard icon is always used). See foreground for colours.

- **fill_colour** (constant)

Colour of fill of folder (only used in the old UI - in the new UI it will be ignored, a standard icon is always used). See foreground for colours.

Returns

no return value

Example

To draw a directory icon on widget btn:

```
btn.DirectoryIcon(Widget.BLACK, Widget.YELLOW);
```

DumpImageString(filename*[string]*, format (optional)*[constant]*)

Description

Dumps a string representation of an image for a widget to a file in a form that can be used by Widget.ReadImageString(). Only possible for Widget.LABEL and Widget.BUTTON widgets.

Arguments

- **filename** (string)

Filename to dump string representation to

- **format (optional)** (constant)

Can be Widget.RGB8 or Widget.RGB24. Before version 15 T/HIS only used 8 bits to store RGB (red, green and blue)

colour information for widget images. In version 15 widget images have been changed to use 24 bits to store RGB information (8 bits for red, 8 bits for green and 8 bits for blue). Both formats are supported. If omitted the new Widget.RGB24 format will be used. See Widget.ReadImageString() for more details.

Returns

no return value

Example

To dump the image data to file 'image_data' for widget w with the old 8 bit RGB representation:

```
w.DumpImageString('image_data', Widget.RGB8);
```

To dump the image data to file 'image_data' for widget w with 24 bit RGB representation:

```
w.DumpImageString('image_data', Widget.RGB24);
```

Hide()

Description

Hides the widget on the screen

Arguments

No arguments

Returns

No return value

Example

To hide widget *w*

```
w.Hide();
```

ItemAt(index*[integer]*)

Description

Returns the WidgetItem object used at *index* in this Widget. See also Widget.TotalItems() and Widget.WidgetItems(). Note that for tree Widgets the items will not be returned in the order that they are displayed in, they will be returned in the order they were added to the tree.

Arguments

- **index** (integer)

index to return WidgetItem for. **Note that indices start at 0**.

Returns

WidgetItem object.

Return type

WidgetItem

Example

To loop over the WidgetItems used in Widget *w*

```
for (i=0; i<w.TotalItems(); i++)
{
    wi = w.ItemAt(i);
}
```

Line(colour*[constant]*, x1*[integer]*, y1*[integer]*, x2*[integer]*, y2*[integer]*)

Description

Draws a line on the widget. Only possible for Widget.LABEL and Widget.BUTTON widgets. The coordinates are local to the Widget, not the Window. See properties xResolution and yResolution for more details. Note that the widget graphics will only be updated when the widget is redrawn. This is to allow the user to do multiple drawing commands on a widget. To force the widget to be redrawn call Show().

Arguments

- **colour** (constant)

Colour of line. See foreground for colours.

- **x1** (integer)

x coordinate of start of line.

- **y1** (integer)

y coordinate of start of line.

- **x2** (integer)

x coordinate of end of line.

- **y2** (integer)

y coordinate of end of line.

Returns

no return value

Example

To draw a red line from (10, 90) to (90, 10) on widget w:

```
w.Line(Widget.RED, 10, 90, 90, 10);
```

MoveWidgetItem(item*[WidgetItem]*, relationship*[constant]*, relitem*[WidgetItem or null]*)

Description

Moves an existing WidgetItem in a tree Widget. Also see Widget.RemoveAllWidgetItems and Widget.RemoveWidgetItem.

Arguments

- **item** (WidgetItem)

WidgetItem to move

- **relationship** (constant)

What relationship (relative to relitem) to use when moving item to the Widget. Can be:
Widget.BEFORE,
Widget.AFTER or
Widget.AFTER.

- **relitem** (WidgetItem or null)

Existing WidgetItem to move item relative to. If relationship is Widget.CHILD then relitem can be null and then the WidgetItem will be moved to the root node of the tree.

Returns

No return value

Example

To move WidgetItem wi in tree widget w after existing WidgetItem ewi:

```
w.MoveWidgetItem(wi, Widget.AFTER, ewi);
```

To move WidgetItem wi in tree widget w as a child of existing WidgetItem ewi:

```
w.MoveWidgetItem(wi, Widget.CHILD, ewi);
```

PixelsPerUnit() [static]

Description

Returns the number of pixels per unit coordinate. This will vary depending on the monitor T/HIS is running on.

Arguments

No arguments

Returns

pixels/unit (real)

Return type

Number

Example

To return how many pixels there are per unit coordinate:

```
var ppu = Widget.PixelsPerUnit();
```

Polygon(colour*[constant]*, fill*[boolean]*, points*[Array of numbers]*)

Description

Draws a polygon on the widget. Only possible for Widget.LABEL and Widget.BUTTON widgets. The coordinates are local to the Widget, not the Window. See properties xResolution and yResolution for more details. Note that the widget graphics will only be updated when the widget is redrawn. This is to allow the user to do multiple drawing commands on a widget. To force the widget to be redrawn call Show(). The number of points (x, y pairs) is limited to 500. Any extra points will be ignored.

Arguments

- **colour** (constant)

Colour of polygon. See foreground for colours.

- **fill** (boolean)

If polygon should be filled or not.

- **points** (Array of numbers)

Array of point coordinates

Returns

no return value

Example

To draw a red filled triangle with corners (20, 20) and (50, 80) and (80, 20) on widget w:

```
var a = new Array(20, 20, 50, 80, 80, 20);
w.Polygon(Widget.RED, true, a);
```

Polygon(colour*[constant]*, fill*[boolean]*, x1*[integer]*, y1*[integer]*, x2*[integer]*, y2*[integer]*, ... xn*[integer]*, ... yn*[integer]*) [deprecated]

This function is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Draws a polygon on the widget. Only possible for Widget.LABEL and Widget.BUTTON widgets. The coordinates are local to the Widget, not the Window. See properties xResolution and yResolution for more details. Note that the widget graphics will only be updated when the widget is redrawn. This is to allow the user to do multiple drawing commands on a widget. To force the widget to be redrawn call Show(). The number of points (x, y pairs) is limited to 500. Any extra points will be ignored.

Arguments

- **colour** (constant)

Colour of polygon. See foreground for colours.

- **fill** (boolean)

If polygon should be filled or not.

- **x1** (integer)

x coordinate of point 1.

- **y1** (integer)

y coordinate of point 1.

- **x2** (integer)

x coordinate of point 2.

- **y2** (integer)

y coordinate of point 2.

- **... xn** (integer)

x coordinate of point n.

- **... yn** (integer)

y coordinate of point n.

Returns

no return value

Example

To draw a red filled triangle with corners (20, 20) and (50, 80) and (80, 20) on widget w:

```
w.Polygon(Widget.RED, true, 20, 20, 50, 80, 80, 20);
```

ReadImageFile(filename*[string]*, justify (optional)*[constant]*, transparent (optional)*[colour value (integer)]*, tolerance (optional)*[integer]*)

Description

Reads an image from a file to show on the widget. Only possible for Widget.LABEL and Widget.BUTTON widgets. The image will be shown on the widget underneath any text. Note that due to the way that colours are used for menus in T/HIS only a small number of colours are available for Widget images. Black and white images will display without any issues but colour images will be displayed with a reduced set of colours.

Arguments

- **filename** (string)

Image file (BMP, GIF, JPEG or PNG) to read. To remove an image use null.

- **justify (optional)** (constant)

Widget justification. Can be a bitwise or of Widget.LEFT, Widget.RIGHT or Widget.CENTRE and Widget.TOP, Widget.MIDDLE or Widget.BOTTOM. Additionally Widget.SCALE can be used to scale the image (either reducing or enlarging it) so that it fills the widget. If omitted the default is Widget.CENTRE|Widget.MIDDLE without scaling.

- **transparent (optional)** (colour value (integer))

Transparent colour. Must be a colour returned by Colour.RGB() in T/HIS. If given then this colour will be replaced by a transparent colour. i.e. the widget background colour will be shown. If omitted or null no transparency will be used.

- **tolerance (optional)** (integer)

Tolerance for transparent colour (0-255).
Any pixels in the image that have a red, green and blue colour value within *tolerance* of the transparent colour will be transparent.
For example if the transparent colour was given as Colour.RGB(255, 0, 0) and *tolerance* is 0 only pixels which have red value 255 **and** green value 0 **and** blue value 0 will be made transparent.
If *tolerance* is 4, pixels which have red values between 251 and 255 **and** green values between 0 and 4 **and** blue values between 0 and 4 will be made transparent.
If omitted a value of 8 will be used.

Returns

no return value

Example

To read image example.png for widget w and place it at the top left:

```
w.ReadImageFile("example.png", Widget.TOP|Widget.LEFT);
```

To read image example.png for widget w and place it at the top left, scaling it to fit the widget:

```
w.ReadImageFile("example.png", Widget.TOP|Widget.LEFT|Widget.SCALE);
```

To read image example.png for widget w and place it at the top left, replacing red with a transparent colour:

```
w.ReadImageFile("example.png", Widget.TOP|Widget.LEFT, Colour.RGB(255, 0, 0));
```

To remove an image from widget w:

```
w.ReadImageFile(null);
```

ReadImageString(string*[string]*, justify (optional)*[constant]*, transparent (optional)*[colour value (integer)]*, tolerance (optional)*[integer]*)

Description

Reads an image from a JavaScript string previously created by Widget.DumpImageString() to show on the widget. Only possible for Widget.LABEL and Widget.BUTTON widgets. The image will be shown on the widget underneath any text.
Note, prior to version 15 of T/HIS only a small number of colours were available for Widget images. In version 14 and earlier the RGB (red, green and blue) information for each pixel in the image was packed into a single byte (8 bits) with 3 bits for red, 3 for green and 2 for blue. Widget.DumpImageString() always returned the string beginning with "RRRGGGBB_RLE" which is this 8 bit format with run length encoding. This is format Widget.RGB8.
In version 15 support for Widget images was enhanced to give 24bit support for colours. The RGB information for each pixel has 8 bits for red, 8 bits for green and 8 bits for blue. This is format Widget.RGB24.
From version 15 Widget.DumpImageString() can either return the the old 8 bit format Widget.RGB8 (string beginning with "RRRGGGBB_RLE") or return the the new 24bit format Widget.RGB24 (string beginning with "RGB24_Z"). ReadImageString supports both formats.

Arguments

- **string** (string)

String containing the image data previously created by Widget.DumpImageString(). To remove an image use null.

- **justify (optional)** (constant)

Widget justification. Can be a bitwise or of Widget.LEFT, Widget.RIGHT or Widget.CENTRE and Widget.TOP, Widget.MIDDLE or Widget.BOTTOM. Additionally Widget.SCALE can be used to scale the image (either reducing or enlarging it) so that it fills the widget. If omitted the default is Widget.CENTRE|Widget.MIDDLE without scaling.

- **transparent (optional)** (colour value (integer))

Transparent colour. Must be a colour returned by Colour.RGB() in T/HIS. If given then this colour will be replaced by a transparent colour. i.e. the widget background colour will be shown. If omitted or null no transparency will be used.

- **tolerance (optional)** (integer)

Tolerance for transparent colour (0-255). Only used for the new 24bit format Widget.RGB24 (strings beginning with "RGB24_Z"). Ignored for the old 8 bit format Widget.RGB8 (strings beginning with "RRRGGGBB_RLE").
Any pixels in the image that have a red, green and blue colour value within *tolerance* of the transparent colour will be transparent.
For example if the transparent colour was given as Colour.RGB(255, 0, 0) and *tolerance* is 0 only pixels which have red value 255 **and** green value 0 **and** blue value 0 will be made transparent.
If *tolerance* is 4, pixels which have red values between 251 and 255 **and** green values between 0 and 4 **and** blue values between 0 and 4 will be made transparent.
If omitted a value of 8 will be used.

Returns

no return value

Example

To read image data from string s for widget w and place it at the top left:

```
w.ReadImageString(s, Widget.TOP|Widget.LEFT);
```

To read image data from string s for widget w and place it at the top left, scaling it to fit the widget:

```
w.ReadImageString(s, Widget.TOP|Widget.LEFT|Widget.SCALE);
```

To read image data from string s for widget w and place it at the top left, replacing red with a transparent colour:

```
w.ReadImageString(s, Widget.TOP|Widget.LEFT, Colour.RGB(255, 0, 0));
```

To remove an image from widget w:

```
w.ReadImageString(null);
```

Rectangle(colour*[constant]*, fill*[boolean]*, x1*[integer]*, y1*[integer]*, x2*[integer]*, y2*[integer]*)

Description

Draws a rectangle on the widget. Only possible for Widget.LABEL and Widget.BUTTON widgets. The coordinates are local to the Widget, not the Window. See properties xResolution and yResolution for more details. Note that the widget graphics will only be updated when the widget is redrawn. This is to allow the user to do multiple drawing commands on a widget. To force the widget to be redrawn call Show().

Arguments

- **colour** (constant)

Colour of rectangle. See foreground for colours.

- **fill** (boolean)

If rectangle should be filled or not.

- **x1** (integer)

x coordinate of first corner of rectangle.

- **y1** (integer)

y coordinate of first corner of rectangle.

- **x2** (integer)

x coordinate of second (opposite) corner of rectangle.

- **y2** (integer)

y coordinate of second (opposite) corner of rectangle.

Returns

no return value

Example

To draw a red filled rectangle with corners (20, 20) and (80, 80) on widget w:

```
w.Rectangle(Widget.RED, true, 20, 20, 80, 80);
```

RemoveAllWidgetItems()

Description

Removes any WidgetItems from the Widget. Also see Widget.AddWidgetItem and Widget.RemoveWidgetItem.

Arguments

No arguments

Returns

No return value

Example

To remove all WidgetItems from widget w:

```
w.RemoveAllWidgetItems();
```

RemoveWidgetItem(item*[WidgetItem]*)

Description

Removes a WidgetItem from the Widget. Also see Widget.AddWidgetItem and Widget.RemoveAllWidgetItems.

Arguments

- **item** (WidgetItem)

WidgetItem to remove

Returns

No return value

Example

To remove WidgetItem wi from widget w:

```
w.RemoveWidgetItem(wi);
```

Scroll(scroll*[constant or WidgetItem object]*)

Description

Scrolls a tree or listbox widget

Arguments

- **scroll** (constant or WidgetItem object)

How to scroll the tree/listbox widget. Can be: Widget.SCROLL_TOP, Widget.SCROLL_BOTTOM, Widget.SCROLL_UP, Widget.SCROLL_DOWN, Widget.SCROLL_PAGE_UP or Widget.SCROLL_PAGE_DOWN in which case the tree/listbox widget will be scrolled by that value or a WidgetItem, in which case the tree/listbox will be scrolled to make the WidgetItem visible, (expanding any branches as necessary to do so for the tree widget).

Returns

No return value

Example

To scroll tree widget w to the top:

```
w.Scroll(Widget.SCROLL_TOP);
```

To scroll listbox widget w so that WidgetItem wi is visible:

```
w.Scroll(wi);
```

ShiftPressed() [static]

Description

Check to see if the Shift key is pressed

Arguments

No arguments

Returns

true/false

Return type

Boolean

Example

To test if someone has the Shift key pressed:

```
if (Widget.ShiftPressed()) { ... }
```

Show()

Description

Shows the widget on the screen

Arguments

No arguments

Returns

No return value

Example

To show widget w:

```
w.Show();
```

Static()

Description

Windows have two different regions for Widgets. A 'normal' region which can be scrolled if required (if the window is made smaller scrollbars will be shown which can be used to scroll the contents) and a 'static' region at the top of the Window which is fixed and does not scroll. For an example of a static region in a Window see any of the keyword editing panels. The 'Dismiss', 'Create', 'Reset' etc buttons are in the static region. By default Widgets are put into the normal region of the Window. This method puts the Widget to the static region of the Window.

Arguments

No arguments

Returns

No return value

Example

To put widget w in the static part of the window:

```
w.Static();
```

StringLength(text*[string]*, monospace (optional)*[boolean]*, fontSize (optional)*[integer]*) [static]

Description

Returns the length of a string in Widget units. This can be used to find what size a Widget must be to be able to display the string.

Arguments

- **text** (string)

Text to find the width of

- **monospace (optional)** (boolean)

If true then width will be calculated using a monospace font. If false (default) then the normal proportional width font will be used

- **fontSize (optional)** (integer)

Calculation can be based on a defined font size, at the moment support is added only for font sizes of 6, 7, 8, 10, 12, 14, 18 and 24.

Returns

integer

Return type

Number

Example

To get the width of string 'Example':

```
var len = Widget.StringLength('Example');
```

Tick(colour (optional)*[constant]*)

Description

Draws a tick symbol on the widget. Only possible for Widget.LABEL and Widget.BUTTON widgets.

Arguments

- **colour (optional)** (constant)

Colour of tick symbol. See foreground for colours. If omitted, current foreground colour is used.

Returns

no return value

Example

To draw a red tick symbol on widget w:

```
w.Tick(Widget.RED);
```

TotalItems()

Description

Returns the number of the WidgetItem objects used in this Widget (or 0 if none used). See also Widget.ItemAt() and Widget.WidgetItems().

Arguments

No arguments

Returns

integer

Return type

Number

Example

To return the total number of WidgetItems used for Widget *w*

```
var total = w.TotalItems();
```

WidgetItems()

Description

Returns an array of the WidgetItem objects used in this Widget (or null if none used). See also Widget.ItemAt() and Widget.TotalItems().

Arguments

No arguments

Returns

Array of WidgetItem objects

Return type

Array

Example

To return WidgetItems used for Widget *w*

```
var wi = w.WidgetItems();
```

WidgetItem class

The WidgetItem class allows you to create items for combobox, listbox, radio button and tree Widgets. More...

The T/HIS JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Member functions

- FirstChild()
- NextSibling()
- Parent()
- PreviousSibling()

WidgetItem properties

Name	Type	Description
background	constant	Widget background colour. Not used for RADIOBUTTON widgets. Can be: Widget.BLACK, Widget.WHITE, Widget.RED, Widget.GREEN, Widget.BLUE, Widget.CYAN, Widget.MAGENTA, Widget.YELLOW, Widget.DARKRED, Widget.DARKGREEN, Widget.DARKBLUE, Widget.GREY, Widget.DARKGREY, Widget.LIGHTGREY or Widget.DEFAULT
expanded	logical	If the widget item is expanded (true) or not (false) in a tree. Only available for widgetitems used in TREE widgets.
foreground	constant	Widget foreground colour. Not used for RADIOBUTTON widgets. Can be: Widget.BLACK, Widget.WHITE, Widget.RED, Widget.GREEN, Widget.BLUE, Widget.CYAN, Widget.MAGENTA, Widget.YELLOW, Widget.DARKRED, Widget.DARKGREEN, Widget.DARKBLUE, Widget.GREY, Widget.DARKGREY, Widget.LIGHTGREY or Widget.DEFAULT
hover	string	WidgetItem's hover text. Not used for RADIOBUTTON widgets.
index (read only)	integer	The index of this widgetitem in the parent widget (undefined if widgetitem is not assigned to a widget).
monospace	boolean	true if the widgetitem uses a monospace font instead of a proportional width font (default). Not used for RADIOBUTTON and TREE widgets.
onClick	function	Function to call when a widget item in a COMBOBOX, LISTBOX or TREE widget is clicked. The Widgetitem object is accessible in the function using the 'this' keyword.
onMouseOver	function	Function to call when the mouse moves over a widget item in a COMBOBOX, LISTBOX or TREE widget. The Widgetitem object is accessible in the function using the 'this' keyword.
selectable	logical	If the widget item can be selected (true) or not (false). Not used for RADIOBUTTON and TREE widgets.
selected	logical	If the widget item is selected (true) or not (false).
text	string	Widget text. If the WidgetItem is used in a tree widget then the tree will not automatically redraw (this is in case you want to change lots of tree nodes at once). In this case, use the Window Redraw method to redraw the window.
visible	logical	If the widget item is visible (true) or not (false) in a tree. A widgetitem will not be visible if it is a child of a widgetitem that is not expanded. Only available for widgetitems used in TREE widgets.

widget (read only)	Widget object	The widget that this item is defined for (null if not set)

Detailed Description

The WidgetItem class allows you to create items for combobox, listbox, radio button and tree Widgets in a Window for a graphical user interface. The following example shows how WidgetItems are used to create a Combobox Widget and how to assign callbacks to determine when the selection has been changed.

```
var items = ["D3PLOT", "PRIMER", "SHELL", "REPORTER", "T/HIS"]
// Create window
var w = new Window("Combobox example", 0.8, 1.0, 0.5, 0.6);
// A simple combobox with a few items
var cl= new Widget(w, Widget.LABEL,   1, 30, 1, 7, "Programs:");
var cb= new Widget(w, Widget.COMBOBOX,  31, 61, 1, 7);
// Add WidgetItems to Combobox
for (i=0; i<items.length; i++)
    var wi = new WidgetItem(cb, items[i]);
// A combobox with many items showing a slider.
var li= new Widget(w, Widget.LABEL,   1, 30, 8, 14, "Long list:");
var ci= new Widget(w, Widget.COMBOBOX,  31, 61, 8, 14);
// Add WidgetItems to Combobox
// As an example we also make some of the WidgetItems unselectable and
// change the background colour
for (i=1; i<=100; i++)
{
    var wi = new WidgetItem(ci, "Item "+i);
    if ( (i % 10) == 5)
    {
        wi.selectable = false;
        wi.background = Widget.WHITE;
    }
}
var e = new Widget(w, Widget.BUTTON,   1, 21, 15, 21, "Exit");
// Assign callbacks
cb.onClick  = clicked;
cb.onChange = changed;
ci.onClick  = clicked;
ci.onChange = changed;
e.onClick   = confirm_exit
// Show the window and start event loop
w.Show();
////////////////////////////////////////////////////////////////////////////
function clicked()
{
// If combobox is clicked then print the current selection
    if (this.selectedItem)
        Message("selection is currently '"+this.selectedItem.text+"'");
}
////////////////////////////////////////////////////////////////////////////
function changed()
{
// If combobox selection is changed then print the new selection
    if (this.selectedItem)
        Message("selection is now '"+this.selectedItem.text+"'");
}
////////////////////////////////////////////////////////////////////////////
function confirm_exit()
{
// Map confirm box
    var ret = Window.Question("Confirm exit", "Are you sure you want to quit?");
// If the user has answered yes then exit from the script.
    if (ret == Window.YES) Exit();
}
```

See the documentation below and the Window and Widget classes for more details.

Constructor

new WidgetItem(widget*[Widget]*, text*[string]*, selectable (optional)*[boolean]*)

Description

Create a new WidgetItem object for a combobox, listbox or radio button widget.

Arguments

- **widget** (Widget)

Combobox, listbox or radio button Widget that the widget item will be created in. This can be null in which case the WidgetItem will be created but not assigned to a Widget. It can be assigned later by using Widget.AddWidgetItem().

- **text** (string)

Text to show on widget item

- **selectable (optional)** (boolean)

If the widget item can be selected. If omitted the widget item will be selectable. Not used for RADIOBUTTON and TREE widgets.

Returns

WidgetItem object

Return type

WidgetItem

new WidgetItem(widget*[Widget]*, text*[string]*, relationship (optional)*[constant]*, relitem (optional)*[WidgetItem]*)

Description

Create a new WidgetItem object for a tree widget. If the widget argument is given and the relationship and relitem arguments are omitted then the widget item will be added as a root node in the tree. If the relationship and relitem arguments are also used then the item can be added at a specific location in the tree. Alternatively, the widget argument can be null, and the relationship and relitem arguments omitted, in which case the WidgetItem will be created but not assigned to a Widget. It can be assigned to the tree later using Widget.AddWidgetItem()

Arguments

- **widget** (Widget)

Tree Widget that the widget item will be created in or null (if the relationship and relitem arguments are omitted)

- **text** (string)

Text to show on widget item

- **relationship (optional)** (constant)

What relationship (relative to relitem) to use when adding item to the Widget. Can be:
Widget.BEFORE,
Widget.AFTER or
Widget.CHILD.

- **relitem (optional)** (WidgetItem)

Existing WidgetItem to add item relative to

Returns

WidgetItem object

Return type

WidgetItem

Details of functions

FirstChild()

Description

Returns the first child WidgetItem or null if the WidgetItem has no children. Only possible for WidgetItems used in Widget.TREE widgets.

Arguments

No arguments

Returns

WidgetItem object

Return type

WidgetItem

Example

To get the first child widget item in a tree widget for widget item wi:

```
var cwi = wi.FirstChild();
```

NextSibling()

Description

Returns the next sibling WidgetItem or null if the WidgetItem has no further siblings. Only possible for WidgetItems used in Widget.TREE widgets.

Arguments

No arguments

Returns

WidgetItem object

Return type

WidgetItem

Example

To get the next sibling widget item in a tree widget for widget item wi:

```
var pwi = wi.NextSibling();
```

Parent()

Description

Returns the parent WidgetItem or null if the WidgetItem has no parent. Only possible for WidgetItems used in Widget.TREE widgets.

Arguments

No arguments

Returns

WidgetItem object

Return type

WidgetItem

Example

To get the parent widget item in a tree widget for widget item wi:

```
var pwi = wi.Parent();
```

PreviousSibling()

Description

Returns the next sibling WidgetItem or null if the WidgetItem has no further siblings. Only possible for WidgetItems used in Widget.TREE widgets.

Arguments

No arguments

Returns

WidgetItem object

Return type

WidgetItem

Example

To get the previous sibling widget item in a tree widget for widget item wi:

```
var pwi = wi.PreviousSibling();
```

Window class

The Window class allows you to create windows for a graphical user interface. More...

The T/HIS JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- BottomBorder()
- BuildGUIFromString(guistring*[string]*)
- EndLoop()
- Error(title*[string]*, error*[string]*, buttons (optional)*[constant]*)
- GetDirectory(initial (optional)*[string]*)
- GetFile(extension (optional)*[string]*, save (optional)*[boolean]*, initial (optional)*[string]*)
- GetFilename(title*[string]*, message*[string]*, extension (optional)*[string]*, initial (optional)*[string]*, save (optional)*[boolean]*)
- GetFiles(extension (optional)*[string]*)
- GetInteger(title*[string]*, message*[string]*, initial (optional)*[integer]*)
- GetNumber(title*[string]*, message*[string]*, initial (optional)*[real]*)
- GetPassword(title*[string]*, message*[string]*)
- GetString(title*[string]*, message*[string]*, initial (optional)*[string]*)
- Information(title*[string]*, info*[string]*, buttons (optional)*[constant]*)
- MasterResolution()
- Message(title*[string]*, message*[string]*, buttons (optional)*[constant]*)
- MiddleBorder()
- Question(title*[string]*, question*[string]*, buttons (optional)*[constant]*)
- RightBorder()
- StartLoop()
- Theme(theme (optional)*[constant]*)
- TopBorder()
- UpdateGUI()
- Warning(title*[string]*, warning*[string]*, buttons (optional)*[constant]*)

Member functions

- Delete()
- Hide()
- Recompute()
- Redraw()
- Show(modal (optional)*[boolean]*)

Window constants

Name	Description
Window.CANCEL	Show CANCEL button
Window.NO	Show NO button
Window.NONMODAL	Allow Window.Error, Window.Question, Window.Warning etc windows to be non modal
Window.OK	Show OK button
Window.YES	Show YES button

Constants for Resizing/positioning

Name	Description

Window.BOTTOM	Bottom resizing/positioning of window
Window.CENTRE	Centre (horizontal) positioning of window
Window.LEFT	Left resizing/positioning of window
Window.MIDDLE	Middle (vertical) positioning of window
Window.REDUCE	Window is allowed to reduce in size when resizing
Window.RIGHT	Right resizing/positioning of window
Window.TOP	Top resizing/positioning of window

Constants for Themes

Name	Description
Window.THEME_CLASSIC	Use the Classic theme (Note: Not only the script will use this theme, the whole interface of the program will switch to classic)
Window.THEME_CURRENT	Use the current theme
Window.THEME_DARK	Use the Dark theme (Note: Not only the script will use this theme, the whole interface of the program will switch to dark)
Window.THEME_LIGHT	Use the Light theme (Note: Not only the script will use this theme, the whole interface of the program will switch to light)
Window.USE_OLD_UI_JS	Use the original, pre v17, theme (default). (Note:The interface of the program will NOT switch to old)

Window properties

Name	Type	Description
active	boolean	If true (default) then the window then the window is active and widgets in the window can be used. If false then the window is inactive and the widgets cannot be used.
background	constant	Window background colour. Can be: Widget.BLACK, Widget.WHITE, Widget.RED, Widget.GREEN, Widget.BLUE, Widget.CYAN, Widget.MAGENTA, Widget.YELLOW, Widget.DARKRED, Widget.DARKGREEN, Widget.DARKBLUE, Widget.GREY, Widget.DARKGREY, Widget.LIGHTGREY or Widget.DEFAULT
bottom	real	bottom coordinate of window in range 0.0 (bottom) to 1.0 (top)
height	real	height of window
keepOnTop	boolean	If true then the window will be kept "on top" of other windows. If false (default) then the window stacking order can be changed.
left	real	left coordinate of window in range 0.0 (left) to 1.0 (right)
maxWidgets (read only)	integer	The maximum number of widgets that can be made in this window. This can be changed **before** the window is created by using Options.max_widgets. Also see totalWidgets
onAfterShow	function	Function to call **after** a Window is shown. The Window object is accessible in the function using the 'this' keyword. This may be useful to ensure that certain actions are done after the window is shown. It can also be used to show another window so this enables multiple windows to be shown. To unset the function set the property to null.
onBeforeShow	function	Function to call **before** a Window is shown. The Window object is accessible in the function using the 'this' keyword. This may be useful to ensure that buttons are shown/hidden etc before the window is shown. Note that it cannot be used to show another window. Use onAfterShow for that. To unset the function set the property to null.

onClose	function	Function to call when a Window is closed by pressing the X on the top right of the window. This may be useful to ensure that certain actions are done when the window is closed. The Window object is accessible in the function using the 'this' keyword. If this function returns false then closing the window will be cancelled. Returning any other value (or not returning anything) will be ignored and the window will close. To unset the function set the property to null.
onHide	function	Function to call when a Window is hidden by calling Hide(). This may be useful to ensure that certain actions are done after the window is hidden. The Window object is accessible in the function using the 'this' keyword. To unset the function set the property to null.
resize	constant	Window resizing. By default when a Window is shown it is allowed to resize on all sides (left, right, top and bottom) to try to make enough room to show the Widgets. The behaviour can be changed by using this property. It can be any combination (bitwise OR) of Window.LEFT, Window.RIGHT, Window.TOP or Window.BOTTOM or 0. In addition Window.REDUCE can also be added to allow the window to reduce in size when resizing. Note that when Window.Show is called this property is set to 0 (i.e. not to resize on any side).
right	real	right coordinate of window in range 0.0 (left) to 1.0 (right)
showClose	boolean	If true (default) then a close (X) button will automatically be added on the top right of the window. If false then no close button will be shown.
shown (read only)	boolean	true if window is currently shown, false if not
title	string	Window title
top	real	top coordinate of window in range 0.0 (bottom) to 1.0 (top)
totalWidgets (read only)	integer	The total number of widgets that have been made in this window. This can be changed **before** the window is created by using Options.max_widgets. Also see maxWidgets
width	real	width of window

Detailed Description

The Window class allows you to make windows that you can place Widgets in to create a graphical user interface. The Widget class also gives a number of static methods for convenience. e.g. Window.GetInteger(). The following very simple example displays some text in a window with a button that unmaps the window when it is pressed and the user confirms that they want to exit.

```
// Create window with title "Text" from 0.8-1.0 in x and 0.5-0.6 in y
var w = new Window("Text", 0.8, 1.0, 0.5, 0.6);
// Create label widget
var l = new Widget(w, Widget.LABEL,   1, 40, 1,  7, "Press OK to exit");
// Create button widget
var e = new Widget(w, Widget.BUTTON, 11, 30, 8, 14, "OK");
// Assign the onClick callback method to the function confirm_exit'
e.onClick  = confirm_exit;
// Show the widget and start event loop
w.Show();
////////////////////////////////////////////////////////////////////////////
function confirm_exit()
{
// Map confirm window
    var ret = Window.Question("Confirm exit", "Are you sure you want to quit?");
// If the user has answered Yes then exit.
    if (ret == Window.YES) Exit();
}
```

See the documentation below and the Widget class for more details.

Constructor

new Window(title*[string]*, left*[real]*, right*[real]*, bottom*[real]*, top*[real]*)

Description

Create a new Window object.

Arguments

- **title** (string)

Window title to show in title bar

- **left** (real)

left coordinate of window in range 0.0 (left) to 1.0 (right)

- **right** (real)

right coordinate of window in range 0.0 (left) to 1.0 (right)

- **bottom** (real)

bottom coordinate of window in range 0.0 (bottom) to 1.0 (top)

- **top** (real)

top coordinate of window in range 0.0 (bottom) to 1.0 (top)

Returns

Window object

Return type

Window

Example

To create a Window 'Example' in the top right half of the screen:

```
var w = new Window('Example', 0.5, 1.0, 0.5, 1.0);
```

Details of functions

BottomBorder() [static]

Description

Returns the vertical position of the bottom border (in range 0-1). This can be used to help position windows on the screen.

Arguments

No arguments

Returns

real in range 0-1

Return type

Number

Example

To obtain the position of the bottom border:

```
var b = Window.BottomBorder();
```

BuildGUIFromString(guistring*[string]*) [static]

Description

Builds a GUI from a JSON string created by the GUI builder.

Arguments

- **guistring** (string)

The string to create the GUI from

Returns

object containing the GUI

Return type

Object

Example

To create a GUI from the string json_string:

```
var gui = Window.BuildGUIFromString(json_string);
```

Delete()

Description

Deletes the window from T/HIS and returns any memory/resources used for the window. **This function should not normally need to be called**. However, in exceptional circumstances if a script recreates windows many times T/HIS may run out of USER objects on Microsoft Windows because of the way T/HIS creates and shows windows. To avoid this problem this method can be used to force T/HIS to return the resources for a window. **Do not use the Window object after calling this method**.

Arguments

No arguments

Returns

No return value

Example

To delete window w:

```
w.Delete();
```

EndLoop() [static]

Description

Ends an event loop started by Window.StartLoop()

Arguments

No arguments

Returns

No return value

Example

To end a blocking event loop started by Window.StartLoop():

```
Window.EndLoop();
```

Error(title*[string]*, error*[string]*, buttons (optional)*[constant]*) [static]

Description

Show an error message in a window.

Arguments

- **title** (string)

Title for window.

- **error** (string)

Error message to show in window. The maximum number of lines that can be shown is controlled by the Options.max_window_lines option.

- **buttons (optional)** (constant)

The buttons to use. Can be bitwise OR of Window.OK, Window.CANCEL, Window.YES or Window.NO. If this is omitted an OK button will be used. By default the window will be modal. If Window.NONMODAL is also given the window will be non-modal instead.

Returns

Button pressed

Return type

Number

Example

To show error *Critical error!\nAbort?* in window with title *Error* with Yes and No buttons:

```
var answer = Window.Error("Error", "Critical error!\nAbort?", Window.YES |
Window.NO);
if (answer == Window.YES) Exit();
```

GetDirectory(initial (optional)*[string]*) [static]

Description

Map the directory selector box native to your machine, allowing you to choose a directory. On Unix this will be a Motif selector. Windows will use the standard windows directory selector.

Arguments

- **initial (optional)** (string)

Initial directory to start from.

Returns

directory (string), (or null if cancel pressed).

Return type

String

Example

To select a directory:

```
var dir = Window.GetDirectory();
```

GetFile(extension (optional)*[string]*, save (optional)*[boolean]*, initial (optional)*[string]*) [static]

Description

Map a file selector box allowing you to choose a file. See also Window.GetFiles() and Window.GetFilename().

Arguments

- **extension (optional)** (string)

Extension to filter by.

- **save (optional)** (boolean)

If true the file selector is to be used for saving a file. If false (default) the file selector is for opening a file. Due to native operating system file selector differences, on linux new filenames can only be given when saving a file. On windows it is possible to give new filenames when opening or saving a file.

- **initial (optional)** (string)

Initial directory to start from.

Returns

filename (string), (or null if cancel pressed).

Return type

String

Example

To select a file using extension '.key':

```
var file = Window.GetFile(".key");
```

GetFilename(title*[string]*, message*[string]*, extension (optional)*[string]*, initial (optional)*[string]*, save (optional)*[boolean]*) [static]

Description

Map a window allowing you to input a filename (or select it using a file selector). OK and Cancel buttons are shown. See also Window.GetFile().

Arguments

- **title** (string)

Title for window.

- **message** (string)

Message to show in window.

- **extension (optional)** (string)

Extension to filter by.

- **initial (optional)** (string)

Initial value.

- **save (optional)** (boolean)

If true the file selector is to be used for saving a file. If false (default) the file selector is for opening a file. Due to native

operating system file selector differences, on linux new filenames can only be given when saving a file. On windows it is possible to give new filenames when opening or saving a file.

Returns

filename (string), (or null if cancel pressed).

Return type

String

Example

To create an file input window with title *Choose file* and message *Choose the file to open* and return the filename input:

```
var filename = Window.GetFilename("Choose file", "Choose the file to open");
```

GetFiles(extension (optional)*[string]*) [static]

Description

Map a file selector box allowing you to choose multiple files. See also Window.GetFile() and Window.GetFilename().

Arguments

- **extension (optional)** (string)

Extension to filter by.

Returns

Array of filenames (strings), or null if cancel pressed.

Return type

String

Example

To select multiple files using extension '.key':

```
var files = Window.GetFiles(".key");
```

GetInteger(title*[string]*, message*[string]*, initial (optional)*[integer]*) [static]

Description

Map a window allowing you to input an integer. OK and Cancel buttons are shown.

Arguments

- **title** (string)

Title for window.

- **message** (string)

Message to show in window.

- **initial (optional)** (integer)

Initial value.

Returns

value input (integer), or null if cancel pressed.

Return type

Number

Example

To create an input window with title *Input* and message *Input integer* and return the value input:

```
var value = Window.GetInteger("Input", "Input integer");
```

GetNumber(title*[string]*, message*[string]*, initial (optional)*[real]*) [static]

Description

Map a window allowing you to input a number. OK and Cancel buttons are shown.

Arguments

- **title** (string)

Title for window.

- **message** (string)

Message to show in window.

- **initial (optional)** (real)

Initial value.

Returns

value input (real), or null if cancel pressed.

Return type

Number

Example

To create an input window with title *Input* and message *Input number* and return the value input:

```
var value = Window.GetNumber("Input", "Input number");
```

GetPassword(title*[string]*, message*[string]*) [static]

Description

Map a window allowing you to input a password. OK and Cancel buttons are shown.
This is identical to Window.GetString except the string is hidden and no initial value can be given.

Arguments

- **title** (string)

Title for window.

- **message** (string)

Message to show in window.

Returns

value input (string), or null if cancel pressed.

Return type

String

Example

To create an input window with title *Input* and message *Input password* and return the value input:

```
var value = Window.GetPassword("Input", "Input password");
```

GetString(title*[string]*, message*[string]*, initial (optional)*[string]*) [static]

Description

Map a window allowing you to input a string. OK and Cancel buttons are shown.

Arguments

- **title** (string)

Title for window.

- **message** (string)

Message to show in window.

- **initial (optional)** (string)

Initial value.

Returns

value input (string), or null if cancel pressed.

Return type

String

Example

To create an input window with title *Input* and message *Input string* and return the value input:

```
var value = Window.GetString("Input", "Input string");
```

Hide()

Description

Hides (unmaps) the window. Also see the Window onHide property.

Arguments

No arguments

Returns

No return value

Example

To hide window w:

```
w.Hide();
```

Information(title*[string]*, info*[string]*, buttons (optional)*[constant]*) [static]

Description

Show information in a window.

Arguments

- **title** (string)

Title for window.

- **info** (string)

Information to show in window. The maximum number of lines that can be shown is controlled by the Options.max_window_lines option.

- **buttons (optional)** (constant)

The buttons to use. Can be bitwise OR of Window.OK, Window.CANCEL, Window.YES or Window.NO. If this is omitted an OK button will be used. By default the window will be modal. If Window.NONMODAL is also given the window will be non-modal instead.

Returns

Button pressed

Return type

Number

Example

To show information *Information* in window with title *Example* with OK and Cancel buttons:

```
var answer = Window.Information("Example", "Information", Window.OK |
Window.CANCEL);
if (answer == Window.CANCEL) Message("You pressed the Cancel button");
```

MasterResolution() [static]

Description

Returns the resolution of the master programme window in pixels

Arguments

No arguments

Returns

Array of numbers containing x and y resolution in pixels

Return type

Number

Example

To get the resolution of the main window:

```
var res = Window.MasterResolution();
```

Message(title*[string]*, message*[string]*, buttons (optional)*[constant]*) [static]

Description

Show a message in a window.

Arguments

- **title** (string)

Title for window.

- **message** (string)

Message to show in window. The maximum number of lines that can be shown is controlled by the Options.max_window_lines option.

- **buttons (optional)** (constant)

The buttons to use. Can be bitwise OR of Window.OK, Window.CANCEL, Window.YES or Window.NO. If this is omitted an OK button will be used By default the window will be modal. If Window.NONMODAL is also given the window will be non-modal instead.

Returns

Button pressed

Return type

Number

Example

To show message *Press YES or NO* in window with title *Example* with YES and NO buttons:

```
var answer = Window.Message("Example", "Press YES or NO", Window.YES |
Window.NO);
if (answer == Window.NO) Message("You pressed No");
```

MiddleBorder() [static]

Description

Returns the vertical position of the middle border (in range 0-1). The middle border is the border between the tools/keywords window and the docked windows. This can be used to help position windows on the screen.

Arguments

No arguments

Returns

real in range 0-1

Return type

Number

Example

To obtain the position of the middle border:

```
var b = Window.MiddleBorder();
```

Question(title*[string]*, question*[string]*, buttons (optional)*[constant]*) [static]

Description

Show a question in a window.

Arguments

- **title** (string)

Title for window.

- **question** (string)

Question to show in window. The maximum number of lines that can be shown is controlled by the Options.max_window_lines option.

- **buttons (optional)** (constant)

The buttons to use. Can be bitwise OR of Window.OK, Window.CANCEL, Window.YES or Window.NO. If this is omitted Yes and No button will be used. By default the window will be modal. If Window.NONMODAL is also given the window will be non-modal instead.

Returns

Button pressed

Return type

Number

Example

To show question *Do you want to continue?* in window with title *Question*:

```
var answer = Window.Question("Question", "Do you want to continue?");
if (answer == Window.NO) Message("You pressed No");
```

Recompute()

Description

Recomputes the positions of widgets in the window. If you have static widgets and 'normal' widgets in a window and you show and/or hide widgets the window needs to be recomputed to refresh the graphics, scroll bars etc. Calling this method will recompute and redraw the window.

Arguments

No arguments

Returns

No return value

Example

To recompute window w:

```
w.Recompute();
```

Redraw()

Description

Redraws the window. Sometimes if you show, hide or draw graphics on widgets the window needs to be redrawn to refresh the graphics. Calling this method will redraw the window refreshing the graphics.

Arguments

No arguments

Returns

No return value

Example

To redraw window w:

```
w.Redraw();
```

RightBorder() [static]

Description

Returns the horizontal position of the right border (in range 0-1). This can be used to help position windows on the screen.

Arguments

No arguments

Returns

real in range 0-1

Return type

Number

Example

To obtain the position of the right border:

```
var b = Window.RightBorder();
```

Show(modal (optional)*[boolean]*)

Description

Shows (maps) the window and waits for user input.

Arguments

- **modal (optional)** (boolean)

If this window is modal (true) then the user is blocked from doing anything else in T/HIS until this window is dismissed). If non-modal (false) then the user can still use other functions in T/HIS.
If omitted the window will be modal.
Note that making a window modal will stop interaction in all other windows and may prevent operations such as picking from working in any macros that are run from scripts.
Before version 21 T/HIS would create a blocking event loop when the Window Show method was called (the call to the Window Show method would not return until the window was hidden again). In T/HIS version 21 the logic has been changed so that Window Show maps the window and returns straight away. The window is managed from the main event loop in T/HIS. In most cases this will make no differences to scripts but there may be rare cases where you specifically want the blocking behaviour. In this case use Window.StartLoop().

Returns

No return value

Example

To show window w:

```
w.Show();
```

To show window w allowing the user to use other functions in T/HIS:

```
w.Show(false);
```

StartLoop() [static]

Description

Starts a blocking event loop in T/HIS. Before version 21 T/HIS would create a blocking event loop when the Window Show method was called (the call to the Window Show method would not return until the window was hidden again). In T/HIS version 21 the logic has been changed so that Window Show maps the window and returns straight away. The window is managed from the main event loop in T/HIS. In most cases this will make no differences to scripts but there may be rare cases where you specifically want the blocking behaviour. An example of this is using a keyout hook script, or if you want to give aprompt/question similarly to Window.Message() where you do **not** want to return to the main event loop. You want to block until the user has specified the action.
Window.StartLoop() can be used to start a local blocking, event loop. To terminate the event loop and resume execution of the script use Window.EndLoop()
Note that only a single loop can be active in T/HIS at any one time. i.e. they cannot be nested. Window.StartLoop() should be used as sparingly as possible and only used for specific short events (e.g. something like the equivalent of Window.Message() where you really do need to block) as if it is used in one script it will prevent it from being used in another script.

Arguments

No arguments

Returns

No return value

Example

To start a blocking event loop:

```
Window.StartLoop();
```

Theme(theme (optional)*[constant]*) [static]

Description

Set or get a user interface theme.

Arguments

- **theme (optional)** (constant)

If it is provided it is used to set the current theme. Can be either Window.USE_OLD_UI_JS, Window.THEME_CURRENT, Window.THEME_DARK, Window.THEME_LIGHT, Window.THEME_CLASSIC.

Returns

Integer. When getting the theme one of: Window.USE_OLD_UI_JS, Window.THEME_DARK, Window.THEME_LIGHT, Window.THEME_CLASSIC

Return type

Number

Example

To determine the current theme:

```
var ui = Window.Theme();
      if(ui == Window.THEME_DARK)
      {
          print("Theme is dark\n");
      }
      else if(ui == Window.THEME_LIGHT)
      {
          print("Theme is light\n");
      }
      else if(ui == Window.THEME_CLASSIC)
      {
          print("Theme is classic\n");
      }
      else
      {
          print("Theme is not set\n");
      }
```

To keep the original (pre v17) appearance of your JavaScript (default):

```
Window.Theme(Window.USE_OLD_UI_JS);
```

To use the current user interface theme:

```
Window.Theme(Window.THEME_CURRENT);
```

To use the dark user interface theme:

```
Window.Theme(Window.THEME_DARK);
```

TopBorder() [static]

Description

Returns the vertical position of the top border (in range 0-1). This can be used to help position windows on the screen. This is no longer used in T/HIS and will always be 1 but is left for backwards compatibility.

Arguments

No arguments

Returns

real in range 0-1

Return type

Number

Example

To obtain the position of the top border:

```
var b = Window.TopBorder();
```

UpdateGUI() [static]

Description

Force GUI to be updated. This function is not normally needed but if you are doing a computationally expensive operation and want to update the GUI it may be necessary as the GUI update requests are cached until there is spare time to update them. Calling this function forces any outstanding requests to be flushed.

Arguments

No arguments

Returns

No return value

Example

To force update of GUI:

```
Window.UpdateGUI();
```

Warning(title*[string]*, warning*[string]*, buttons (optional)*[constant]*) [static]

Description

Show a warning message in a window.

Arguments

- **title** (string)

Title for window.

- **warning** (string)

Warning message to show in window. The maximum number of lines that can be shown is controlled by the Options.max_window_lines option.

- **buttons (optional)** (constant)

The buttons to use. Can be bitwise OR of Window.OK, Window.CANCEL, Window.YES or Window.NO. If this is omitted an OK button will be used. By default the window will be modal. If Window.NONMODAL is also given the window will be non-modal instead.

Returns

Button pressed

Return type

Number

Example

To show warning *Title is blank\nSet to ID?* in window with title *Warning* with Yes and No buttons:

```
var answer = Window.Warning("Warning", "Title is blank\nSet to ID?", Window.YES
| Window.NO);
if (answer == Window.NO) Message("You pressed No");
```

Workflow class

The Workflow class allows you to read and write workflow files. More...

The T/HIS JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- GetWorkflowDataFromPath(folder*[string]*, workflow_name (optional)*[string]*)
- ModelIdFromIndex(model_index*[integer]*, workflow_name (optional)*[string]*)
- ModelUnitSystemFromIndex(model_index*[integer]*, workflow_name (optional)*[string]*)
- ModelUserDataBuildFromIndex(model_index*[integer]*, workflow_name (optional)*[string]*)
- ModelUserDataFromIndex(model_index*[integer]*, workflow_name (optional)*[string]*)
- ModelUserDataProgramFromIndex(model_index*[integer]*, workflow_name (optional)*[string]*)
- ModelUserDataVersionFromIndex(model_index*[integer]*, workflow_name (optional)*[string]*)
- NumberOfSelectedModels(workflow_name (optional)*[string]*)
- Refresh()
- WorkflowDefinitionFilename(workflow_name (optional)*[string]*)
- WriteToFile(user_data*[object]*, output_filename*[string]*, workflow_definition_filename*[string]*, extra (optional)*[object]*)

Workflow constants

Constants for UnitSystem

Name	Description
Workflow.UNIT_SYSTEM_NONE	No unit system
Workflow.UNIT_SYSTEM_U1	U1 unit system (m, kg, s)
Workflow.UNIT_SYSTEM_U2	U2 unit system (mm, t, s)
Workflow.UNIT_SYSTEM_U3	U3 unit system (mm, kg, ms)
Workflow.UNIT_SYSTEM_U4	U4 unit system (mm, g, ms)
Workflow.UNIT_SYSTEM_U5	U5 unit system (ft, slug, s)
Workflow.UNIT_SYSTEM_U6	U6 unit system (m, t, s)

Detailed Description

The Workflow class gives you simple functions to read and write workflow files

Details of functions

GetWorkflowDataFromPath(folder*[string]*, workflow_name (optional)*[string]*) [static]

Description

Returns user data in the specified folder. This will first search for user data in *.json files in the specified folder and upwards in parent folders if there is no user data in the specified folder. The number of folders it will search is controlled by the preference oasys*workflow_max_upward_folder_search_depth. Additionally, if the preference oasys*workflow_user_directory_name is defined it will search in the folder with that name too in the specified folder and parent folders. Secondly, it will search for user data in keyword files in the specified folder.

Arguments

- **folder** (string)

The folder to search for user data.

- **workflow_name (optional)** (string)

The workflow name to return the user data for. If not specified it will return user data for all the workflows found.

Returns

Array of objects with the following properties:

Name	Type	Description
data	object	User data as an object
model_unit_system	integer	Model unit system associated with the user data
workflow_data_filename	string	Filename where workflow data was read from
workflow_definition	string	Workflow definition filename
workflow_name	string	Workflow name

Return type

object

ModelIdFromIndex(model_index*[integer]*, workflow_name (optional)*[string]*) [static]

Description

Returns the id of a model selected by the user by index (starting at 0).

Arguments

- **model_index** (integer)

The index of the model to return the ID for. If the workflow is run from the workflow menu and the name argument is not defined, it is the index in the list of models selected by the user. If the workflow is run from the workflow menu and the name argument is defined, it is the index of the model that has user data for the named workflow, out of the list of models selected by the user. If the workflow is run from REPORTER, it is the index in the list of all the models loaded in the session that have user data for the named workflow.

- **workflow_name (optional)** (string)

The workflow name to return the model id for. If it is not specified it uses the name of the workflow selected by the user from the workflow menu.

Returns

integer

Return type

Number

Example

To get the id of the first model selected by the user

```
var id = Workflow.ModelIdFromIndex(0);
```

To get the id of the second model and workflow name "Automotive Assessment"

```
var id = Workflow.ModelIdFromIndex(1, "Automotive Assessment");
```

ModelUnitSystemFromIndex(model_index*[integer]*, workflow_name (optional)*[string]*) [static]

Description

Returns the unit system of a model selected by the user by index (starting at 0). Will be Workflow.UNIT_SYSTEM_NONE or Workflow.UNIT_SYSTEM_U1 or Workflow.UNIT_SYSTEM_U2 or Workflow.UNIT_SYSTEM_U3 or Workflow.UNIT_SYSTEM_U4 or Workflow.UNIT_SYSTEM_U5 or Workflow.UNIT_SYSTEM_U6

Arguments

- **model_index** (integer)

The index of the model to return the unit system for. If the workflow is run from the workflow menu and the name argument is not defined, it is the index in the list of models selected by the user. If the workflow is run from the workflow menu and the name argument is defined, it is the index of the model that has user data for the named workflow, out of the list of models selected by the user. If the workflow is run from REPORTER, it is the index in the list of all the models loaded in the session that have user data for the named workflow.

- **workflow_name (optional)** (string)

The workflow name to return the unit system for.

Returns

integer

Return type

Number

Example

To get the unit system for the workflow selected from the menu, for the first model selected by the user

```
var unit_system = Workflow.ModelUnitSystemFromIndex(0);
```

To get the unit system of the second model and workflow name "Automotive Assessment"

```
var unit_system = Workflow.ModelUnitSystemFromIndex(1, "Automotive Assessment");
```

ModelUserDataBuildFromIndex(model_index*[integer]*, workflow_name (optional)*[string]*) [static]

Description

Returns the build number of the program that was used to write out the user data of a model for the selected workflow by index (starting at 0).

Arguments

- **model_index** (integer)

The index of the model to return the build number for. If the workflow is run from the workflow menu and the name argument is not defined, it is the index in the list of models selected by the user. If the workflow is run from the workflow menu and the name argument is defined, it is the index of the model that has user data for the named workflow, out of the list of models selected by the user. If the workflow is run from REPORTER, it is the index in the list of all the models loaded in the session that have user data for the named workflow.

- **workflow_name (optional)** (string)

The workflow name to return the build number for.

Returns

number

Return type

number

Example

To get the build number of the program that was used to write user data for the first model that has data for the workflow selected by the user

```
var build = Workflow.ModelUserDataBuildFromIndex(0);
```

To get the build number of the program that was used to write user data for the second model that has data for the "Automotive Assessment" workflow

```
var build = Workflow.ModelUserDataBuildFromIndex(1, "Automotive Assessment");
```

ModelUserDataFromIndex(model_index*[integer]*, workflow_name (optional)*[string]*) [static]

Description

Returns the user data associated with a model that has data for the selected workflow by index (starting at 0).

Arguments

- **model_index** (integer)

The index of the model to return the user data for. If the workflow is run from the workflow menu and the name argument is not defined, it is the index in the list of models selected by the user. If the workflow is run from the workflow menu and the name argument is defined, it is the index of the model that has user data for the named workflow, out of the list of models selected by the user. If the workflow is run from REPORTER, it is the index in the list of all the models loaded in the session that have user data for the named workflow.

- **workflow_name (optional)** (string)

The workflow name to return the user data for. If it is not specified it uses the name of the workflow selected by the user from the workflow menu.

Returns

object

Return type

Object

Example

To get the user data for the first model that has data for the workflow selected by the user

```
var user_data = Workflow.ModelUserDataFromIndex(0);
```

To get the user data for the second model that has data for the "Automotive Assessment" workflow

```
var user_data = Workflow.ModelUserDataFromIndex(1, "Automotive Assessment");
```

ModelUserDataProgramFromIndex(model_index*[integer]*, workflow_name (optional)*[string]*) [static]

Description

Returns the name of the program that was used to write out the user data of a model for the selected workflow by index (starting at 0).

Arguments

- **model_index** (integer)

The index of the model to return the program name for. If the workflow is run from the workflow menu and the name argument is not defined, it is the index in the list of models selected by the user. If the workflow is run from the workflow menu and the name argument is defined, it is the index of the model that has user data for the named workflow, out of the list of models selected by the user. If the workflow is run from REPORTER, it is the index in the list of all the models loaded in the session that have user data for the named workflow.

- **workflow_name (optional)** (string)

The workflow name to return the program name for.

Returns

string

Return type

String

Example

To get the name of the program that was used to write user data for the first model that has data for the workflow selected by the user

```
var program = Workflow.ModelUserDataProgramFromIndex(0);
```

To get the name of the program that was used to write user data for the second model that has data for the "Automotive Assessment" workflow

```
var program = Workflow.ModelUserDataProgramFromIndex(1, "Automotive
Assessment");
```

ModelUserDataVersionFromIndex(model_index*[integer]*, workflow_name (optional)*[string]*) [static]

Description

Returns the version of the program that was used to write out the user data of a model for the selected workflow by index (starting at 0).

Arguments

- **model_index** (integer)

The index of the model to return the program version for. If the workflow is run from the workflow menu and the name argument is not defined, it is the index in the list of models selected by the user. If the workflow is run from the workflow menu and the name argument is defined, it is the index of the model that has user data for the named workflow, out of the list of models selected by the user. If the workflow is run from REPORTER, it is the index in the list of all the models loaded in the session that have user data for the named workflow.

- **workflow_name (optional)** (string)

The workflow name to return the program version for.

Returns

number

Return type

number

Example

To get the version of the program that was used to write user data for the first model that has data for the workflow selected by the user

```
var version = Workflow.ModelUserDataVersionFromIndex(0);
```

To get the version of the program that was used to write user data for the second model that has data for the "Automotive Assessment" workflow

```
var version = Workflow.ModelUserDataVersionFromIndex(1, "Automotive
Assessment");
```

NumberOfSelectedModels(workflow_name (optional)*[string]*) [static]

Description

Returns the number of models selected by the user.

Arguments

- **workflow_name (optional)** (string)

The workflow name to return the number of models for. If it’s not defined the number of models that were selected by the user on the workflow menu is returned. If it’s defined and the workflow was run from the workflow menu, the number of models, out of the models selected by the user, that have data for the named workflow is returned. If it’s defined and the workflow is run from REPORTER, the number of models, out of all the models loaded in the session, that have data for the named workflow is returned.

Returns

integer

Return type

Number

Example

To get the number of models selected by the user in the workflow menu

```
var n = Workflow.NumberOfSelectedModels();
```

To get the number of models that have user data for the "Automotive Assessment" workflow

```
var n = Workflow.NumberOfSelectedModels("Automotive Assessment");
```

Refresh() [static]

Description

Scan for fresh workflow data

Arguments

No arguments

Returns

No return value

Example

```
Workflow.Refresh();
```

WorkflowDefinitionFilename(workflow_name (optional)*[string]*) [static]

Description

Returns the workflow definition filename

Arguments

- **workflow_name (optional)** (string)

The workflow name to return the workflow defintion filename for. This is required when a POST item is generated from REPORTER. If it is not specified the first workflow user data associated with the model is returned (this is the 'normal' case where a user launches a workflow from the workflow menu).

Returns

string

Return type

String

Example

To get the worklow definition filename that the script has been run with

```
var workflow_definition_filename = Workflow.WorkflowDefinitionFilename();
```

WriteToFile(user_data*[object]*, output_filename*[string]*, workflow_definition_filename*[string]*, extra (optional)*[object]*) [static]

Description

Writes a workflow to a JSON file. If the file already exists the workflow is added to the file (or updated if it is already in the file).

Arguments

- **user_data** (object)

Object containing user data required for the workflow.

- **output_filename** (string)

Filename to write to.

- **workflow_definition_filename** (string)

Filename of the workflow definition file.

- **extra (optional)** (object)

Extra workflow information

Object has the following properties:

Name	Type	Description
model_unit_system (optional)	integer	The model unit system. Can be Workflow.UNIT_SYSTEM_NONE or Workflow.UNIT_SYSTEM_U1 or Workflow.UNIT_SYSTEM_U2 or Workflow.UNIT_SYSTEM_U3 or Workflow.UNIT_SYSTEM_U4 or Workflow.UNIT_SYSTEM_U5 or Workflow.UNIT_SYSTEM_U6

Returns

No return value

Example

To write the file "C:\my_workflow.json" with some user data and the contents of the workflow definition file "C:\workflows\my_workflow_definition.json"

```
      var user_data = { part_ids: [1, 2, 10, 100], node_ids: [12, 23, 24] };
      Workflow.WriteToFile(user_data, "C:\\my_workflow.json",
"C:\\workflows\\my_workflow_definition.json");
```

XMLParser class

The XMLParser class enables reading data from XML files. More...

The T/HIS JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Member functions

- Parse(filename*[string]*)

XMLParser properties

Name	Type	Description
characterDataHandler	function	Function to call when character data is found. The function will be called with 1 argument which is a string containing the character data
commentHandler	function	Function to call when a comment is found. The function will be called with 1 argument which is a string containing the text inside the comment
endCDATAHandler	function	Function to call at the end of a CDATA section. The function does not have any arguments.
endElementHandler	function	Function to call when an element end tag is found. The function will be called with 1 argument which is a string containing the name of the element
startCDATAHandler	function	Function to call at the start of a CDATA section. The function does not have any arguments.
startElementHandler	function	Function to call when an element start tag is found. The function will be called with 2 arguments. Argument 1 is a string containing the name of the element. Argument 2 is an object containing the element attributes

Detailed Description

The XMLParser class provides a stream-oriented parser to enable you to read XML files. You register callback (or handler) functions with the parser and then parse the document. As the parser recognizes parts of the document, it will call the appropriate handler for that part (if you've registered one.) The document is fed to the parser in pieces. This allows you to parse really huge documents that won't fit into memory.
There are currently 6 handlers which can be set: XMLParser.startElementHandler, XMLParser.endElementHandler, XMLParser.characterDataHandler, XMLParser.commentHandler, XMLParser.startCDATAHandler and XMLParser.endCDATAHandler.
The following simple example shows how the parser could be used.

```
// Create a new parser object
var p = new XMLParser();
// assign handlers
p.startElementHandler  = startElem;
p.endElementHandler    = endElem;
p.characterDataHandler = text;
p.commentHandler       = comment;
// parse the file
p.Parse("/data/test.xml");
////////////////////////////////////////////////////////////////////////
function startElem(name, attr)
{
// handler to be called when a start element is found
// Print element name
```

```
    Println("START: " + name);
// Print attributes
    for (n in attr)
    {
        Println(" attr: " + n + "=" + attr[n]);
    }
}
function endElem(name)
{
// handler to be called when an end element is found
// Print element name
    Println("END: " + name);
}
function text(str)
{
// handler to be called when text is found
// Print text
    Println("TEXT: '" + str + "'");
}
function comment(str)
{
// handler to be called when a comment is found
// Print comment
    Println("COMMENT: '" + str + "'");
}
```

See the documentation below for more details.

Constructor

new XMLParser()

Description

Create a new XMLParser object for reading XML files.

Arguments

No arguments

Returns

XMLParser object

Return type

XMLParser

Example

To create a new XMLParser object

```
var p = new XMLParser();
```

Details of functions

Parse(filename*[string]*)

Description

starts parsing an XML file

Arguments

- **filename** (string)

XML file to parse

Returns

No return value

Example

To parse XML file "/data/test.xml"

```
var p = new XMLParser();
p.Parse("/data/test.xml");
```

global class

The global class is the root object in Javascript. More...

The REPORTER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Batch()
- Debug(string*[Any valid javascript type]*)
- Exit()
- GetCurrentDirectory()
- LogError(message*[Any valid javascript type]*)
- LogPrint(message*[Any valid javascript type]*)
- LogWarning(message*[Any valid javascript type]*)
- Output(string*[Any valid javascript type]*)
- SetCurrentDirectory(directory*[string]*)
- System(string*[Any valid javascript type]*)
- Unix()
- Windows()

global properties

Detailed Description

When Reporter is started a **single** global class object is created. All of the standard JavaScript functions and properties are available from it.
In addition an instance of a Reporter class is available, from the global reporter property. The reporter object allows you to access the properties and templates used in Reporter.

Details of functions

Batch() [static]

Description

This method can be used to test whether REPORTER is running in batch mode or not.

Arguments

No arguments

Returns

true/false

Return type

Boolean

Example

To check if REPORTER is running in batch mode

```
if (Batch()) { do something }
```

Debug(string*[Any valid javascript type]*) [static]

Description

Print a string to log file for debugging. Anything that you call the debug method on will be 'printed' to the log file window. **Note that a carriage return will automatically be added**.

Arguments

- **string** (Any valid javascript type)

The string/item that you want to debug

Returns

No return value

Example

To print string "Hello, world!" to the debug log file

```
Debug("Hello, world!");
```

Exit() [static]

Description

Stop execution and exit from script

Arguments

No arguments

Returns

No return value

Example

Exit from script with

```
Exit();
```

GetCurrentDirectory() [static]

Description

Return the current working directory for REPORTER.

Arguments

No arguments

Returns

string

Return type

String

Example

To return the current directory

```
var dir = GetCurrentDirectory();
```

LogError(message*[Any valid javascript type]*) [static]

Description

Print an error to log file. Anything that you print will be output to the log file window in bold red text. **Note that a carriage return will automatically be added**.

Arguments

- **message** (Any valid javascript type)

The string/item that you want to print

Returns

No return value

Example

To give error "Error: something has gone wrong" to the log file

```
LogError("Error: something has gone wrong");
```

LogPrint(message*[Any valid javascript type]*) [static]

Description

Print a string to log file. Anything that you print will be output to the log file window. **Note that a carriage return will automatically be added**.

Arguments

- **message** (Any valid javascript type)

The string/item that you want to print

Returns

No return value

Example

To print string "Hello, world!" to the log file

```
LogPrint("Hello, world!");
```

LogWarning(message*[Any valid javascript type]*) [static]

Description

Print a warning to log file. Anything that you print will be output to the log file window in red text. **Note that a carriage return will automatically be added**.

Arguments

- **message** (Any valid javascript type)

The string/item that you want to print

Returns

No return value

Example

To give warning "Warning: something has gone wrong" to the log file

```
LogWarning("Warning: something has gone wrong");
```

Output(string*[Any valid javascript type]*) [static]

Description

Output a string from a script. **Note that a carriage return is not automatically added**.

Arguments

- **string** (Any valid javascript type)

The string/item that you want to print

Returns

No return value

Example

To output string "Hello, world!" with a carriage return:

```
Output("Hello, world!\n");
```

SetCurrentDirectory(directory*[string]*) [static]

Description

Set the current working directory for REPORTER.

Arguments

- **directory** (string)

The directory that you want to change to

Returns

true if successful, false if not

Return type

Boolean

Example

To set the current directory to C:\temp

```
var status = SetCurrentDirectory("C:\\temp");
```

System(string*[Any valid javascript type]*) [static]

Description

Do a system command outside REPORTER.

Arguments

- **string** (Any valid javascript type)

The system command that you want to do

Returns

integer (probably zero if command successful but is implementation-dependant)

Return type

Number

Example

To make the directory "example"

```
System("mkdir example");
```

Unix() [static]

Description

Test whether script is running on a Unix/Linux operating system. See also Windows()

Arguments

No arguments

Returns

true if Unix/Linux, false if not

Return type

Boolean

Example

To test if the OS is Unix

```
if ( Unix() )
```

Windows() [static]

Description

Test whether script is running on a Windows operating system. See also Unix()

Arguments

No arguments

Returns

true if Windows, false if not

Return type

Boolean

Example

To test if the OS is Windows

```
if ( Windows() )
```

Colour class

The Colour class gives access to colours in Reporter. More...

The REPORTER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Black()
- Blue()
- Cyan()
- Green()
- Grey10()
- Grey20()
- Grey30()
- Grey40()
- Grey50()
- Grey60()
- Grey70()
- Grey80()
- Grey90()
- Magenta()
- None()
- RGB(red*[integer]*, green*[integer]*, blue*[integer]*)
- Red()
- White()
- Yellow()

Colour properties

Name	Type	Description
blue (read only)	integer	Colour blue component (0-255)
green (read only)	integer	Colour green component (0-255)
name (read only)	string	Colour name
red (read only)	integer	Colour red component (0-255)

Detailed Description

The Colour class is used to define colours, either by predefined colours or by RGB values. The easiest way to set the colour of something is to use the predefined colour methods. e.g. to set the text colour of item i to red:

```
i.textColour = Colour.Red();
```

For other colours use Colour.RGB().

Details of functions

Black() [static]

Description

Creates a black colour

Arguments

No arguments

Returns

Colour object

Return type

Colour

Example

To set the text colour of item i to black:

```
i.textColour = Colour.Black();
```

Blue() [static]

Description

Creates a blue colour

Arguments

No arguments

Returns

Colour object

Return type

Colour

Example

To set the text colour of item i to blue:

```
i.textColour = Colour.Blue();
```

Cyan() [static]

Description

Creates a cyan colour

Arguments

No arguments

Returns

Colour object

Return type

Colour

Example

To set the text colour of item i to cyan:

```
i.textColour = Colour.Cyan();
```

Green() [static]

Description

Creates a green colour

Arguments

No arguments

Returns

Colour object

Return type

Colour

Example

To set the text colour of item i to green:

```
i.textColour = Colour.Green();
```

Grey10() [static]

Description

Creates a 10% grey colour

Arguments

No arguments

Returns

Colour object

Return type

Colour

Example

To set the text colour of item i to 10% grey:

```
i.textColour = Colour.Grey10();
```

Grey20() [static]

Description

Creates a 20% grey colour

Arguments

No arguments

Returns

Colour object

Return type

Colour

Example

To set the text colour of item i to 10% grey:

```
i.textColour = Colour.Grey20();
```

Grey30() [static]

Description

Creates a 30% grey colour

Arguments

No arguments

Returns

Colour object

Return type

Colour

Example

To set the text colour of item i to 30% grey:

```
i.textColour = Colour.Grey30();
```

Grey40() [static]

Description

Creates a 40% grey colour

Arguments

No arguments

Returns

Colour object

Return type

Colour

Example

To set the text colour of item i to 40% grey:

```
i.textColour = Colour.Grey40();
```

Grey50() [static]

Description

Creates a 50% grey colour

Arguments

No arguments

Returns

Colour object

Return type

Colour

Example

To set the text colour of item i to 50% grey:

```
i.textColour = Colour.Grey50();
```

Grey60() [static]

Description

Creates a 60% grey colour

Arguments

No arguments

Returns

Colour object

Return type

Colour

Example

To set the text colour of item i to 60% grey:

```
i.textColour = Colour.Grey60();
```

Grey70() [static]

Description

Creates a 70% grey colour

Arguments

No arguments

Returns

Colour object

Return type

Colour

Example

To set the text colour of item i to 70% grey:

```
i.textColour = Colour.Grey70();
```

Grey80() [static]

Description

Creates a 80% grey colour

Arguments

No arguments

Returns

Colour object

Return type

Colour

Example

To set the text colour of item i to 80% grey:

```
i.textColour = Colour.Grey80();
```

Grey90() [static]

Description

Creates a 90% grey colour

Arguments

No arguments

Returns

Colour object

Return type

Colour

Example

To set the text colour of item i to 90% grey:

```
i.textColour = Colour.Grey90();
```

Magenta() [static]

Description

Creates a magenta colour

Arguments

No arguments

Returns

Colour object

Return type

Colour

Example

To set the text colour of item i to magenta:

```
i.textColour = Colour.Magenta();
```

None() [static]

Description

No colour

Arguments

No arguments

Returns

Colour object

Return type

Colour

Example

To set the fill colour of item i to no colour:

```
i.fillColour = Colour.None();
```

RGB(red*[integer]*, green*[integer]*, blue*[integer]*) [static]

Description

Creates a colour from red, green and blue components

Arguments

- **red** (integer)

red component of colour (0-255).

- **green** (integer)

green component of colour (0-255).

- **blue** (integer)

blue component of colour (0-255).

Returns

Colour object

Return type

Colour

Example

To set the text colour of item i to red:

```
i.textColour = Colour.RGB(255, 0, 0);
```

Red() [static]

Description

Creates a red colour

Arguments

No arguments

Returns

Colour object

Return type

Colour

Example

To set the text colour of item i to red:

```
i.textColour = Colour.Red();
```

White() [static]

Description

Creates a white colour

Arguments

No arguments

Returns

Colour object

Return type

Colour

Example

To set the text colour of item i to white:

```
i.textColour = Colour.White();
```

Yellow() [static]

Description

Creates a yellow colour

Arguments

No arguments

Returns

Colour object

Return type

Colour

Example

To set the text colour of item i to yellow:

```
i.textColour = Colour.Yellow();
```

File class

The File class allows you to read and write from text files. More...

The REPORTER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- ConvertSeparators(filename*[string]*)
- Copy(source*[string]*, dest*[string]*)
- Delete(filename*[string]*)
- Directory(filename*[string]*)
- DriveMapFilename(filename*[string]*, format*[constant]*)
- Exists(filename*[string]*)
- FindFiles(directory*[string]*, pattern*[string]*, recursive*[boolean]*)
- IsAbsolute(filename*[string]*)
- IsDirectory(filename*[string]*)
- IsFile(filename*[string]*)
- Mkdir(name*[string]*)
- Move(source*[string]*, dest*[string]*)
- SimplifyName(filename*[string]*)
- Size(filename*[string]*)

Member functions

- Close()
- FindLineContaining(contain*[string]*)
- FindLineMatching(regex*[RegExp]*)
- FindLineStarting(start*[string]*)
- Flush()
- ReadChar()
- ReadLine()
- ReadLongLine()
- Seek(position*[integer]*)
- Write(string*[Any valid javascript type]*)

File constants

Name	Description
File.APPEND	Flag to open file for appending
File.EOF	Flag to indicate end of file
File.READ	Flag to open file for reading
File.WRITE	Flag to open file for writing

Detailed Description

The File class allows you to read text and write text to files. There are various functions available that allow to to find lines matching specific strings or regular expressions when reading.
Additionally, there are a number of utility functions to check if a file exists or is a directory etc.

Constructor

new File(filename*[string]*, mode*[constant]*)

Description

Create a new File object for reading and writing text files.

Arguments

- **filename** (string)

Filename of the file you want to read/write. If reading, the file must exist. If writing, the file will be overwritten if it already exists

- **mode** (constant)

The mode to open the file with. Can be File.READ, File.WRITE or File.APPEND

Returns

File object

Return type

File

Example

To create a new file object to read file "/data/test/file.txt"

```
var f = new File("/data/test/file.txt", File.READ);
```

Details of functions

Close()

Description

Close a file opened by a File object.

Arguments

No arguments

Returns

No return value

Example

To close File object f.

```
f.Close();
```

ConvertSeparators(filename*[string]*) [static]

Description

Convert directory separators to the correct type for this operating system

Arguments

- **filename** (string)

Filename you want to convert separators on.

Returns

string filename

Return type

String

Example

e.g. on windows the filename "c:/test/file.key" would be converted to "c:\test\file.key" by

```
var converted = File.ConvertSeparators("c:/test/file.key");
```

Copy(source*[string]*, dest*[string]*) [static]

Description

Copy a file

Arguments

- **source** (string)

Source filename you want to copy.

- **dest** (string)

Destination filename you want to copy source file to. Note that if a file with the name dest already exists it will not be overwritten. Delete the file first with File.Delete().

Returns

true if copy successful, false otherwise.

Return type

Boolean

Example

To copy the file "/data/test/file.txt" to "/data/test/file.txt_backup"

```
var copied = File.Copy("/data/test/file.txt", "/data/test/file.txt_backup");
```

Delete(filename*[string]*) [static]

Description

Delete a file

Arguments

- **filename** (string)

Filename you want to delete.

Returns

true if successful, false if not

Return type

Boolean

Example

To delete the file "/data/test/file.txt"

```
var deleted = File.Delete("/data/test/file.txt");
```

Directory(filename*[string]*) [static]

Description

Extract directory name from an absolute filename

Arguments

- **filename** (string)

Absolute filename you want to extract directory from.

Returns

string directory

Return type

String

Example

To extract the directory "/data/test/" from file "/data/test/file.key"

```
var directory = File.Directory("/data/test/file.key");
```

DriveMapFilename(filename*[string]*, format*[constant]*) [static]

Description

Changes a filename or directory name to the correct format for a specific operating system using the directory mappings (if present).

Arguments

- **filename** (string)

Filename you want to drive map.

- **format** (constant)

The format for the file/directory name. Can be Include.NATIVE, Include.UNIX, or Include.WINDOWS.

Returns

string containing drive mapped filename.

Return type

String

Example

If REPORTER has drive S:\ mapped to /data/ (by using the oasys*drive_s preference)

```
var mapped = File.DriveMapFilename("/data/test/file.ptf", Include.WINDOWS);
```

mapped will be "S:\test\file.ptf".

```
var mapped = File.DriveMapFilename("S:\\test\\file.ptf", Include.UNIX);
```

mapped will be "/data/test/file.ptf",

Exists(filename*[string]*) [static]

Description

Check if a file exists

Arguments

- **filename** (string)

Filename you want to check for existance.

Returns

true/false

Return type

Boolean

Example

To see if the file "/data/test/file.key" exists

```
if (File.Exists("/data/test/file.key")) { do something }
```

FindFiles(directory*[string]*, pattern*[string]*, recursive*[boolean]*) [static]

Description

Find any files in a directory (and subdirectories if required) matching a pattern

Arguments

- **directory** (string)

Directory to look for files in

- **pattern** (string)

Pattern to use to find matching files

- **recursive** (boolean)

If Reporter should look for files recursively or not

Returns

array filenames

Return type

String

Example

To find all of the files matching the pattern "*.key" recursively from directory /data/test

```
var filelist = File.FindFiles("/data/test/", "*.key", true);
```

FindLineContaining(contain*[string]*)

Description

Reads a line from a file which contains contain, opened for reading by a File object. To enable this function to be as fast as possible a maximum line length of 256 characters is used. If you expect a file to have lines longer than 256 characters then use ReadLongLine which allows lines of any length. If one argument is used then the line must contain that string. If more than one argument is used then lines which contain any of the arguments will be returned

Arguments

- **contain** (string)

String which matching lines must contain (maximum length of 256 characters).

This argument can be repeated if required
Alternatively a single array argument containing the multiple values can be given

Returns

string read from file or `File.EOF` if end of file

Return type

String

Example

Loop, reading lines from File object f which contain 'example'.

```
var line;
while ( (line = file.FindLineContaining("example") ) != File.EOF)
{
}
```

FindLineMatching(regex*[RegExp]*)

Description

Reads a line from a file opened for reading by a File object. To enable this function to be as fast as possible a maximum line length of 256 characters is used. If you expect a file to have lines longer than 256 characters then use ReadLongLine which allows lines of any length. Note that this may be much slower than FindLineStarting or FindLineContaining, especially if the regular expression is very complicated.

Arguments

- **regex** (RegExp)

Regular expression which matching lines must match with.

Returns

string read from file or `File.EOF` if end of file

Return type

String

Example

Loop, reading lines from File object f which contain digits.

```
var line;
var regex = new RegExp("\\d+");
while ( (line = file.FindLineMatching(regex) ) != File.EOF)
{
}
```

FindLineStarting(start*[string]*)

Description

Reads a line from a file which starts with start, opened for reading by a File object. To enable this function to be as fast as possible a maximum line length of 256 characters is used. If you expect a file to have lines longer than 256 characters then use ReadLongLine which allows lines of any length. If one argument is used then the line must start with that string. If more than one argument is used then lines which start with any of the arguments will be returned

Arguments

- **start** (string)

String which matching lines must start with (maximum length of 256 characters).

This argument can be repeated if required
Alternatively a single array argument containing the multiple values can be given

Returns

string read from file or `File.EOF` if end of file

Return type

String

Example

Loop, reading lines from File object f which start 'example'.

```
var line;
while ( (line = file.FindLineStarting("example") ) != File.EOF)
{
}
```

Flush()

Description

Flushes a file opened for writing by a File object.

Arguments

No arguments

Returns

No return value

Example

To flush File object f.

```
f.Flush();
```

IsAbsolute(filename*[string]*) [static]

Description

Check if a filename is absolute

Arguments

- **filename** (string)

Filename you want to test if absolute.

Returns

true/false

Return type

Boolean

Example

To see if the file "/data/test/file.key" is absolute

```
if (File.IsAbsolute("/data/test/file.key")) { do something }
```

IsDirectory(filename*[string]*) [static]

Description

Check if a filename is a directory

Arguments

- **filename** (string)

Filename you want to test to see if it is a directory.

Returns

true/false

Return type

Boolean

Example

To see if "/data/test" is a directory

```
if (File.IsDirectory("/data/test")) { do something }
```

IsFile(filename*[string]*) [static]

Description

Check if a filename is a file

Arguments

- **filename** (string)

Filename you want to test to see if it is a file (i.e. not a directory).

Returns

true/false

Return type

Boolean

Example

To see if "/data/test" is a file

```
if (File.IsFile("/data/test")) { do something }
```

Mkdir(name*[string]*) [static]

Description

makes a directory

Arguments

- **name** (string)

Directory you want to create.

Returns

true if successful

Return type

Boolean

Example

To make directory "/data/test" if it does not exist:

```
if (!File.IsDirectory("/data/test")) File.Mkdir("/data/test");
```

Move(source*[string]*, dest*[string]*) [static]

Description

Move a file

Arguments

- **source** (string)

Source filename you want to move.

- **dest** (string)

Destination filename you want to move (rename) source file to. Note that if a file with the name dest already exists it will not be overwritten. Delete the file first with File.Delete().

Returns

true if move successful, false otherwise.

Return type

Boolean

Example

To move the file "/data/test/file.txt" to "/data/test/file.txt_backup"

```
var moved = File.Move("/data/test/file.txt", "/data/test/file.txt_backup");
```

ReadChar()

Description

Reads a single character from a file opened for reading by a File object.

Arguments

No arguments

Returns

character read from file or `File.EOF` if end of file

Return type

String

Example

Loop, reading characters from File object f.

```
 var c;
while ( (c = f.ReadChar()) != undefined) { ... }
```

ReadLine()

Description

Reads a line from a file opened for reading by a File object. To enable this function to be as fast as possible a maximum line length of 256 characters is used. If you expect a file to have lines longer than 256 characters then use ReadLongLine which allows lines of any length.

Arguments

No arguments

Returns

string read from file or `File.EOF` if end of file

Return type

String

Example

Loop, reading lines from File object f.

```
var line;
while ( (line = file.ReadLine() ) != File.EOF)
{
}
```

ReadLongLine()

Description

Reads a line from a file opened for reading by a File object. The line can be any length. If your file has lines shorter than 256 characters then you may want to use ReadLine instead which is faster.

Arguments

No arguments

Returns

string read from file or `File.EOF` if end of file

Return type

String

Example

Loop, reading lines from File object f.

```
var line;
while ( (line = file.ReadLongLine() ) != File.EOF)
{
}
```

Seek(position*[integer]*)

Description

Sets the file position for reading a file

Arguments

- **position** (integer)

Position you want to seek to.

Returns

No return value

Example

To seek to position 1000 in file object f:

```
f.Seek(1000);
```

SimplifyName(filename*[string]*) [static]

Description

Simplify the name of a file by removing //, /./ and /../

Arguments

- **filename** (string)

Filename you want to simplify.

Returns

string filename

Return type

String

Example

To simplify the filename "/data/test//../file.key"

```
var simple = File.SimplifyName("/data/test//../file.key");
```

This simplifies to "/data/file.key"

Size(filename*[string]*) [static]

Description

Get the size of a file

Arguments

- **filename** (string)

File you want to find the size of.

Returns

integer

Return type

Number

Example

To find the size of file "/data/test"

```
var size = File.Size("/data/test");
```

Write(string*[Any valid javascript type]*)

Description

Write a string to a file opened for writing by a File object

Arguments

- **string** (Any valid javascript type)

The string/item that you want to write

Returns

No return value

Example

To write string "Hello, world!" to File object f

```
f.Write("Hello, world!\n");
```

To write the title of model 2 to File object f

```
f.Write("The title of model 2 is " + models[2].title + "\n");
```

Image class

The Image class allows you to create bitmaps in Reporter. More...

The REPORTER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Member functions

- Ellipse(x1*[integer]*, y1*[integer]*, x2*[integer]*, y2*[integer]*)
- Fill(x*[integer]*, y*[integer]*, tol (optional)*[integer]*)
- Line(x1*[integer]*, y1*[integer]*, x2*[integer]*, y2*[integer]*)
- Load(filename*[string]*)
- PixelCount(colour*[string]*, tol (optional)*[integer]*)
- Polygon(points*[array]*)
- Polygon(x1*[integer]*, y1*[integer]*, x2*[integer]*, y2*[integer]*, ... xn*[integer]*, ... yn*[integer]*) [deprecated]
- Polyline(points*[array]*)
- Polyline(x1*[integer]*, y1*[integer]*, x2*[integer]*, y2*[integer]*, ... xn*[integer]*, ... yn*[integer]*) [deprecated]
- Rectangle(x1*[integer]*, y1*[integer]*, x2*[integer]*, y2*[integer]*)
- Save(filename*[string]*, filetype*[constant]*)
- Star(x*[integer]*, y*[integer]*, r*[integer]*)
- Text(x*[integer]*, y*[integer]*, text*[string]*)

Image constants

Name	Description
Image.BMP	Save image as BMP
Image.JPG	Save image as JPG
Image.PNG	Save image as PNG

Image properties

Name	Type	Description
antialiasing	bool	Whether or not lines, shapes and text are drawn with antialiasing (true by default).
fillColour	string	Colour to use when filling shapes on the Image. Can be "none", a valid colour from the X colour database (For Linux users, see /etc/X11/rgb.txt) e.g. "Blue", or #RRGGBB (each of R, G and B is a single hex digit) e.g. "#0000FF" for blue.
font	string	Font to use when drawing text on the Image e.g. "Courier". Can be any font accessible by REPORTER.
fontAngle	integer	Angle (degrees) text is drawn at on the Image. Can be between -360 and 360 degrees.
fontColour	string	Colour to use when drawing text on the Image. Can be "none", a valid colour from the X colour database (For Linux users, see /etc/X11/rgb.txt) e.g. "Blue", or #RRGGBB (each of R, G and B is a single hex digit) e.g. "#0000FF" for blue.
fontJustify	constant	Justification to use when drawing text on the Image. Can be Reporter.JUSTIFY_CENTRE, Reporter.JUSTIFY_LEFT or Reporter.JUSTIFY_RIGHT
fontSize	integer	Size of font (in points) to use when drawing text on the Image

fontStyle	constant	Style of font to use when drawing text on the Image. Can be any combination of Reporter.TEXT_NORMAL, Reporter.TEXT_BOLD, Reporter.TEXT_ITALIC and Reporter.TEXT_UNDERLINE
height	integer	Height of the Image
lineCapStyle	constant	Style to use for the end of lines on an Image. Can be Reporter.CAP_FLAT, Reporter.CAP_SQUARE or Reporter.CAP_ROUND
lineColour	string	Colour to use when drawing lines on the Image. Can be "none", a valid colour from the X colour database (For Linux users, see /etc/X11/rgb.txt) e.g. "Blue", or #RRGGBB (each of R, G and B is a single hex digit) e.g. "#0000FF" for blue.
lineJoinStyle	constant	Style to use for the line join at vertices of polygons and polylines on an Image. Can be Reporter.JOIN_MITRE, Reporter.JOIN_BEVEL or Reporter.JOIN_ROUND
lineStyle	constant	Style to use when drawing lines on an Image. Can be Reporter.LINE_NONE, Reporter.LINE_SOLID, Reporter.LINE_DASH, Reporter.LINE_DOT, Reporter.LINE_DASH_DOT or Reporter.LINE_DASH_DOT_DOT
lineWidth	integer	Width to use when drawing lines on an Image value
width	integer	Width of the Image

Detailed Description

The Image class allows you to create, load and save bitmaps. There are various functions available that allow to to draw lines, rectangles, ellipses, text etc on a bitmap.

Constructor

new Image(width*[integer]*, height*[integer]*, backgroundColour (optional)*[string]*)

Description

Create a new Image object for creating an image. If only 2 arguments are given they are used as the width and height of the image. The third argument can be used to define the initial background colour (the default is white).

Arguments

- **width** (integer)

Width of image

- **height** (integer)

Height of image

- **backgroundColour (optional)** (string)

Initial background colour for the image (default is white). Can be "none", a valid colour from the X colour database (For Linux users, see /etc/X11/rgb.txt) e.g. "Blue", or #RRGGBB (each of R, G and B is a single hex digit) e.g. "#0000FF" for blue.

Returns

Image object

Return type

Image

Example

To create a new image object 100 pixels wide by 50 pixels high

```
var img = new Image(100, 50);
```

Details of functions

Ellipse(x1*[integer]*, y1*[integer]*, x2*[integer]*, y2*[integer]*)

Description

Draw an ellipse on an image

Arguments

- **x1** (integer)

X coordinate of start position for ellipse

- **y1** (integer)

Y coordinate of start position for ellipse

- **x2** (integer)

X coordinate of end position for ellipse

- **y2** (integer)

Y coordinate of end position for ellipse

Returns

no return value

Example

To draw an ellipse with no fill and solid red border line width 2 pixels, on image ’idata’, starting at point 30, 20 and finishing at point 100, 50

```
idata.lineColour = "red"; idata.fillColour = "none";
idata.lineWidth = 2;
idata.lineStyle = Reporter.LINE_SOLID;
idata.Ellipse(30, 20, 100, 50);
```

Fill(x*[integer]*, y*[integer]*, tol (optional)*[integer]*)

Description

Fill an area in an image with a colour.

Arguments

- **x** (integer)

X coordinate of start position for fill

- **y** (integer)

Y coordinate of start position for fill

- **tol (optional)** (integer)

Tolerance for colour matching (0-255). Default is 0. When filling a shape if the red, green and blue components are within tol of the colour of pixel (x, y) the pixel will be filled with the current fill colour.

Returns

no return value

Example

To fill an area of image ’idata’, starting at point 30, 20 with red:

```
idata.fillColour = "red";
idata.Fill(30, 20);
```

Line(x1*[integer]*, y1*[integer]*, x2*[integer]*, y2*[integer]*)

Description

Draw a line on an image

Arguments

- **x1** (integer)

X coordinate of start position for line

- **y1** (integer)

Y coordinate of start position for line

- **x2** (integer)

X coordinate of end position for line

- **y2** (integer)

Y coordinate of end position for line

Returns

no return value

Example

To draw a blue, dashed line width 2 pixels, on image 'idata', starting at point 30, 20 and finishing at point 100, 50

```
idata.lineColour = "blue"; idata.lineWidth = 2; idata.lineStyle = Reporter.LINE_
DASH; idata.Line(30, 20, 100, 50);
```

Load(filename*[string]*)

Description

Load an image file (gif, png, bmp or jpeg)

Arguments

- **filename** (string)

Imagename you want to load.

Returns

no return value

Example

To load the image file "/data/test/image.jpg" into the image object 'idata'

```
idata.Load("/data/test/image.jpg");
```

PixelCount(colour*[string]*, tol (optional)*[integer]*)

Description

Count the number of pixels in an image that have a specific colour.

Arguments

- **colour** (string)

A valid colour from the X colour database (For Linux users, see /etc/X11/rgb.txt) e.g. "Blue", or #RRGGBB (each of R, G and B is a single hex digit) e.g. "#0000FF" for blue

- **tol (optional)** (integer)

Tolerance for colour matching (0-255). Default is 0. When looking at pixels if the red, green and blue components are within tol of the colour of pixel (x, y) the pixel will be counted.

Returns

Number of pixels (integer) with the colour.

Return type

Number

Example

To count the number of red pixels in image 'idata':

```
var nred = idata.PixelCount("red");
```

Polygon(points*[array]*)

Description

Draw a polygon on an image. The last point is always connected back to the first point.

Arguments

- **points** (array)

Array of point coordinates

Returns

no return value

Example

To draw a blue polygon with a solid red border line width 2 pixels, on image 'idata', connecting points (10,10) (20,10) (20,20) (10,20)

```
idata.fillColour = "blue"; idata.lineColour = "red"; idata.lineWidth =
2; idata.lineStyle = Reporter.LINE_SOLID; var a = new Array(10,10,
20,10, 20,20, 10,20); idata.Polygon(a);
```

Polygon(x1*[integer]*, y1*[integer]*, x2*[integer]*, y2*[integer]*, ... xn*[integer]*, ... yn*[integer]*) [deprecated]

This function is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Draw a polygon on an image. The last point is always connected back to the first point.

Arguments

- **x1** (integer)

X coordinate of point 1

- **y1** (integer)

Y coordinate of point 1

- **x2** (integer)

X coordinate of point 2

- **y2** (integer)

Y coordinate of point 2

- **... xn** (integer)

X coordinate of point n

- **... yn** (integer)

Y coordinate of point n

Returns

no return value

Example

To draw a blue polygon with a solid red border line width 2 pixels, on image 'idata', connecting points (10,10) (20,10) (20,20) (10,20)

```
idata.fillColour = "blue"; idata.lineColour = "red"; idata.lineWidth = 2;
idata.lineStyle = Reporter.LINE_SOLID;
idata.Polygon(10,10, 20,10, 20,20, 10,20);
```

Polyline(points*[array]*)

Description

Draw a line with multiple straight segments on an image

Arguments

- **points** (array)

Array of point coordinates

Returns

no return value

Example

To draw a blue, dashed polyline width 2 pixels, on image 'idata', connecting points (10,10) (20,10) (20,20) (10,20)

```
idata.lineColour = "blue"; idata.lineWidth = 2; idata.lineStyle = Reporter.LINE_
DASH; var a = new Array(10,10, 20,10, 20,20, 10,20); idata.Polyline(a);
```

Polyline(x1*[integer]*, y1*[integer]*, x2*[integer]*, y2*[integer]*, ... xn*[integer]*, ... yn*[integer]*) [deprecated]

This function is deprecated in version 21.0. It is only provided to keep old scripts working. We strongly advise against using it in new scripts. Support may be removed in future versions.

Description

Draw a line with multiple straight segments on an image

Arguments

- **x1** (integer)

X coordinate of point 1

- **y1** (integer)

Y coordinate of point 1

- **x2** (integer)

X coordinate of point 2

- **y2** (integer)

Y coordinate of point 2

- **... xn** (integer)

X coordinate of point n

- **... yn** (integer)

Y coordinate of point n

Returns

no return value

Example

To draw a blue, dashed polyline width 2 pixels, on image 'idata', connecting points (10,10) (20,10) (20,20) (10,20)

```
idata.lineColour = "blue"; idata.lineWidth = 2; idata.lineStyle = Reporter.LINE_
DASH; idata.Polyline(10,10, 20,10, 20,20, 10,20);
```

Rectangle(x1*[integer]*, y1*[integer]*, x2*[integer]*, y2*[integer]*)

Description

Draw a rectangle on an image

Arguments

- **x1** (integer)

X coordinate of start position for rectangle

- **y1** (integer)

Y coordinate of start position for rectangle

- **x2** (integer)

X coordinate of end position for rectangle

- **y2** (integer)

Y coordinate of end position for rectangle

Returns

no return value

Example

To draw a rectangle with no fill and solid red border line width 2 pixels, on image 'idata', starting at point 30, 20 and finishing at point 100, 50

```
idata.lineColour = "red"; idata.fillColour = "none";
idata.lineWidth = 2;
idata.lineStyle = Reporter.LINE_SOLID;
idata.Rectangle(30, 20, 100, 50);
```

Save(filename*[string]*, filetype*[constant]*)

Description

Save an image to file (png, bmp or jpeg)

Arguments

- **filename** (string)

Imagename you want to save.

- **filetype** (constant)

Type you want to save as. Can be: Image.BMP, Image.JPG or Image.PNG

Returns

no return value

Example

To save the image object ’idata’ to file "/data/test/image.jpg" as a jpeg

```
idata.Save("/data/test/image.jpg", Image.JPG);
```

Star(x*[integer]*, y*[integer]*, r*[integer]*)

Description

Draw a star on an image

Arguments

- **x** (integer)

X coordinate of centre of star

- **y** (integer)

Y coordinate of centre of star

- **r** (integer)

Radius of star

Returns

no return value

Example

To draw a blue star with yellow fill, on image ’idata’, centred at point 30, 20 with radius 10

```
idata.lineColour = "blue";
idata.fillColour = "yellow";
idata.Star(30, 20, 10);
```

Text(x*[integer]*, y*[integer]*, text*[string]*)

Description

Draw text on an image

Arguments

- **x** (integer)

X position for text

- **y** (integer)

Y position for text

- **text** (string)

Text to write on image

Returns

no return value

Example

To write the text ’Test’ in Helvetica 12pt bold underlined, coloured red on image ’idata’, at point 30, 20

```
idata.fontColour = "red";
idata.fontSize = 12;
idata.fontStyle = Reporter.TEXT_BOLD | Reporter.TEXT_UNDERLINE; idata.Text(30,
20, "Test");
```

Include class

The Include class allows you to access the include files in a model. More...

The REPORTER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Include constants

Constants for Directory separators

Name	Description
Include.NATIVE	Use directory separators native to this machine when writing directory names.
Include.UNIX	Use unix directory separators when writing directory names.
Include.WINDOWS	Use windows directory separators when writing directory names.

Detailed Description

Originally developed for use in PRIMER, the Include class allows a user to create and query include files in a model. A stripped-back version of this class has been added to T/HIS and REPORTER for consistency between the programs.
See File.DriveMapFilename for the current use of this class in REPORTER.

Item class

The Item class gives access to items in Reporter. More...

The REPORTER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- GetAll(page*[Page]*)
- GetFromName(page*[Page]*, name*[string]*)

Member functions

- DeleteColumn(column*[integer]*)
- DeleteRow(row*[integer]*)
- Generate()
- GetCellProperties(row*[integer]*, column*[integer]*)
- GetColumnProperties(column*[integer]*, header*[constant]*)
- GetColumnWidth(row*[integer]*)
- GetCondition(index*[integer]*)
- GetCondition(index*[integer]*, column*[integer]*)
- GetCondition(index*[integer]*, row*[integer]*, column*[integer]*)
- GetGeneratedData(row_index*[integer]*, column_index*[integer]*)
- GetRowHeight(row*[integer]*)
- InsertColumn(column*[integer]*)
- InsertRow(row*[integer]*)
- MergeCells(topLeftRow*[integer]*, topLeftColumn*[integer]*, rows*[integer]*, columns*[integer]*)
- RemoveCondition(condition*[integer]*)
- RemoveCondition(condition*[integer]*, column*[integer]*)
- RemoveCondition(condition*[integer]*, row*[integer]*, column*[integer]*)
- SetCellProperties(properties*[object]*, row*[integer]*, column*[integer]*)
- SetColumnProperties(properties*[object]*, column*[integer]*, header*[constant]*)
- SetColumnWidth(column*[integer]*, width*[real]*)
- SetCondition(condition*[integer]*, properties*[object]*)
- SetCondition(condition*[integer]*, column*[integer]*, properties*[object]*)
- SetCondition(condition*[integer]*, row*[integer]*, column*[integer]*, properties*[object]*)
- SetRowHeight(row*[integer]*, height*[real]*)
- UnmergeCells(row*[integer]*, column*[integer]*)

Item constants

Name	Description
Item.ARROW	Arrow item
Item.AUTO_TABLE	Automatic table item
Item.D3PLOT	D3PLOT item
Item.ELLIPSE	Ellipse item
Item.IMAGE	Image item
Item.IMAGE_FILE	Image file item
Item.LIBRARY_IMAGE	Library image item
Item.LIBRARY_PROGRAM	Library program item

Item.LINE	Line item
Item.NOTE	Note item
Item.PLACEHOLDER	Placeholder item
Item.PRIMER	PRIMER item
Item.PROGRAM	Program item
Item.RECTANGLE	Rectangle item
Item.SCRIPT	Script item
Item.SCRIPT_FILE	Script File item
Item.TABLE	Table item
Item.TEXT	Text item
Item.TEXTBOX	Textbox item
Item.TEXT_FILE	Text file item
Item.THIS	T/HIS item

Item properties

Name	Type	Description
active	logical	If item is active or not. Inactive items will be skipped during report/page/item generation.
autotableType	constant	Autotable type (whether the data is sourced from a file or a directory). Can be Reporter.AUTO_TABLE_DIRECTORY or Reporter.AUTO_TABLE_FILE. Valid for item type Item.AUTO_TABLE.
bottomCrop	integer	Bottom cropping value. Valid for item types Item.IMAGE, Item.IMAGE_FILE, Item.D3PLOT, Item.PRIMER and Item.THIS.
bottomMargin	real	Bottom margin width. Valid for item types Item.TEXTBOX, Item.TEXT_FILE, Item.TABLE and Item.AUTO_TABLE
columns (read only)	integer	The number of columns in the table. Valid for item types Item.TABLE and Item.AUTO_TABLE
conditions (read only)	integer	The number of conditions assigned to the item. Valid for item types Item.PROGRAM, Item.TEXT_FILE, Item.TEXT and Item.TEXTBOX
embed	logical	If image is embedded or not. Valid for item types Item.IMAGE
file	string	File or directory for item. Valid for item types: Item.AUTO_TABLE Item.D3PLOT Item.IMAGE Item.IMAGE_FILE Item.PRIMER Item.PROGRAM Item.SCRIPT_FILE Item.TEXT_FILE Item.THIS
filetype (read only)	string	Output file type. Read-only but can be updated by changing the file extension on the item property "file". Valid for item types Item.D3PLOT Item.PRIMER and Item.THIS.
fillColour	Colour object	Colour of fill for the item. Valid for item types Item.RECTANGLE, Item.ELLIPSE, Item.TEXTBOX, Item.PROGRAM and Item.TEXT_FILE
fontName	string	Font for the item e.g. "Courier". Can be any font accessible by REPORTER. Valid for item types Item.TEXT, Item.TEXTBOX, Item.PROGRAM and Item.TEXT_FILE

fontSize	integer	Font size for the item (6 <= fontSize <= 72). Valid for item types Item.TEXT, Item.TEXTBOX, Item.PROGRAM and Item.TEXT_FILE
fontStyle	constant	Font style for the item. Can be a combination of Reporter.TEXT_NORMAL, Reporter.TEXT_BOLD, Reporter.TEXT_ITALIC or Reporter.TEXT_UNDERLINE Valid for item types Item.TEXT, Item.TEXTBOX, Item.PROGRAM and Item.TEXT_FILE
generatedRowHeight	real	The height of each generated row in an Autotable. Valid for item type Item.AUTO_TABLE.
generating (read only)	logical	true if the item is currently being generated
headerHeight	real	The height of the header in an Autotable. Valid for item type Item.AUTO_TABLE.
height	real	Height for "rectangular" items (absolute difference between y and y2). Note that depending on what your object reference corner preference is set to, changing this property will result in different behaviour.
job	string	Input job file. Valid for item types Item.D3PLOT and Item.THIS.
justify	constant	Text justification for the item. Can be Reporter.JUSTIFY_CENTRE, Reporter.JUSTIFY_LEFT or Reporter.JUSTIFY_RIGHT combined with Reporter.JUSTIFY_TOP, Reporter.JUSTIFY_MIDDLE or Reporter.JUSTIFY_BOTTOM Valid for item types Item.TEXT, Item.TEXTBOX, Item.PROGRAM and Item.TEXT_FILE
leftCrop	integer	Left cropping value. Valid for item types Item.IMAGE, Item.IMAGE_FILE, Item.D3PLOT, Item.PRIMER and Item.THIS.
leftMargin	real	Left margin width. Valid for item types Item.TEXTBOX, Item.TEXT_FILE, Item.TABLE and Item.AUTO_TABLE
lineColour	Colour object	Colour of outline for the item. Valid for item types Item.LINE, Item.ARROW, Item.RECTANGLE, Item.ELLIPSE, Item.TEXTBOX, Item.D3PLOT, Item.PRIMER, Item.THIS, Item.PROGRAM, Item.TEXT_FILE, Item.IMAGE_FILE, Item.TABLE and Item.AUTO_TABLE.
lineStyle	constant	Style of outline for the item. Can be Reporter.LINE_NONE, Reporter.LINE_SOLID, Reporter.LINE_DASH, Reporter.LINE_DOT, Reporter.LINE_DASH_DOT or Reporter.LINE_DASH_DOT_DOT Valid for item types Item.LINE, Item.ARROW, Item.RECTANGLE, Item.ELLIPSE, Item.TEXTBOX, Item.D3PLOT, Item.PRIMER, Item.THIS, Item.PROGRAM, Item.TEXT_FILE and Item.IMAGE_FILE.
lineWidth	real	Width of outline for the item in mm. Valid for item types Item.LINE, Item.ARROW, Item.RECTANGLE, Item.ELLIPSE, Item.TEXTBOX, Item.D3PLOT, Item.PRIMER, Item.THIS, Item.PROGRAM, Item.TEXT_FILE, Item.IMAGE_FILE, Item.TABLE and Item.AUTO_TABLE
name	string	Name of the Item
resolution	integer	Image resolution. Larger values yield a smaller image representation on screen. Valid for item type Item.IMAGE.
rightCrop	integer	Right cropping value. Valid for item types Item.IMAGE, Item.IMAGE_FILE, Item.D3PLOT, Item.PRIMER and Item.THIS.
rightMargin	real	Right margin width. Valid for item types Item.TEXTBOX, Item.TEXT_FILE, Item.TABLE and Item.AUTO_TABLE
rows (read only)	integer	The number of rows in the table. Valid for item type Item.TABLE

saveCSV	bool	Whether or not a CSV file of the table contents is written when the item is generated. Valid for item types Item.TABLE and Item.AUTO_TABLE
saveCSVFilename	string	The path and filename of the CSV file written when the item is generated. Valid for item types Item.TABLE and Item.AUTO_TABLE
saveXlsx	bool	Whether or not a Excel file of the table contents is written when the item is generated. Valid for item types Item.TABLE and Item.AUTO_TABLE
saveXlsxFilename	string	The path and filename of the Excel file written when the item is generated. Valid for item types Item.TABLE and Item.AUTO_TABLE
script	string	The script source text for the item. Only valid for item type Item.SCRIPT. For Item.SCRIPT_FILE, use the *file* property.
text	string	The text for the item. Valid for item types Item.TEXT, Item.TEXTBOX, Item.PROGRAM, Item.TEXT_FILE and Item.SCRIPT
textColour	Colour object	Colour of text for the item. Valid for item types Item.TEXT, Item.TEXTBOX, Item.PROGRAM and Item.TEXT_FILE
topCrop	integer	Top cropping value. Valid for item types Item.IMAGE, Item.IMAGE_FILE, Item.D3PLOT, Item.PRIMER and Item.THIS.
topMargin	real	Top margin width. Valid for item types Item.TEXTBOX, Item.TEXT_FILE, Item.TABLE and Item.AUTO_TABLE
type (read only)	constant	type of the Item. Can be Item.LINE, Item.TEXT etc.
width	real	Width for "rectangular" items (absolute difference between x and x2). Note that depending on what your object reference corner preference is set to, changing this property will result in different behaviour.
x	real	X coordinate
x2	real	Second X coordinate for "rectangular" items
y	real	Y coordinate
y2	real	Second Y coordinate for "rectangular" items

Detailed Description

The Item class allows you to access the items in templates that Reporter currently has open.

Constructor

new Item(page*[Page]*, type*[constant]*, name (optional)*[string]*, x (optional)*[real]*, x2 (optional)*[real]*, y (optional)*[real]*, y2 (optional)*[real]*)

Description

Create a new Item. The name and coordinates arguments are optional. Item.TABLE items are constructed with two rows and two columns by default. If you require only one row or column, use DeleteRow and DeleteColumn.

Arguments

- **page** (Page)

Page to create item in

- **type** (constant)

Item type. Can be Item.LINE, Item.ARROW, Item.RECTANGLE, Item.ELLIPSE, Item.TEXT, Item.TEXTBOX, Item.IMAGE, Item.PROGRAM, Item.D3PLOT, Item.PRIMER, Item.THIS, Item.TEXT_FILE, Item.IMAGE_FILE, Item.LIBRARY_IMAGE, Item.LIBRARY_PROGRAM, Item.TABLE, Item.AUTO_TABLE, Item.SCRIPT, Item.SCRIPT_FILE, Item.NOTE or Item.PLACEHOLDER.

- **name (optional)** (string)

Name of item

- **x (optional)** (real)

X coordinate

- **x2 (optional)** (real)

Second X coordinate for "rectangular" items

- **y (optional)** (real)

Y coordinate

- **y2 (optional)** (real)

Second Y coordinate for "rectangular" items

Returns

Item object

Return type

Item

Example

To create a new blank Item object:

```
var i = new Item();
```

Details of functions

DeleteColumn(column*[integer]*)

Description

Delete a column from a table. Valid for item type Item.TABLE and Item.AUTO_TABLE.

Arguments

- **column** (integer)

The index of the column to delete. Note that indices start from 0.

Returns

No return value

Example

To delete the second column from table item i:

```
i.DeleteColumn(1);
```

DeleteRow(row*[integer]*)

Description

Delete a row from a table. Valid for item type Item.TABLE.

Arguments

- **row** (integer)

The index of the row to delete. Note that indices start from 0.

Returns

No return value

Example

To delete the second row from table item i:

```
i.DeleteRow(1);
```

Generate()

Description

Generate an item.

Arguments

No arguments

Returns

No return value

Example

To generate item i:

```
i.Generate();
```

GetAll(page*[Page]*) [static]

Description

Get all of the items on a page.

Arguments

- **page** (Page)

Page to get items from.

Returns

Array of Item objects

Return type

Array

Example

To get all of the items on page p:

```
var items = Item.GetAll(p);
```

GetCellProperties(row*[integer]*, column*[integer]*)

Description

Get the properties of the specified cell. Valid for item type Item.TABLE.

Arguments

- **row** (integer)

The row index of the cell of interest. Note that indices start from 0.

- **column** (integer)

The column index of the cell of interest. Note that indices start from 0.

Returns

Object with the following properties:

Name	Type	Description
bottomBorderWidth	real	Cell bottom border width. Can be 0.0, 0.1, 0.5, 0.75, 1.0, 1.5, 2.0, 2.5, 3.0, 4.0, 5.0. Other values will result in no border.
colspan	integer	Number of columns this cell spans (for merged cells). 1 if not merged. Use columnMergeOrigin to find top-left cell.
column	integer	The column index
columnMergeOrigin	integer	The column index of the top-left cell in this merge cell group (if cell not merged then == column).
conditions	integer	Number of conditions assigned to this cell.
fillColour	Colour object	Fill colour
fontName	string	Font name (e.g. "Courier").
fontSize	integer	Font size (between 6 and 72).
fontStyle	integer	Font style. See Text style constants for details.
height	real	Cell height. Modifying this property will modify the height of all cells in the row.
hyperlinkHTML	string	Hyperlink destination for HTML.
hyperlinkPDF	string	Hyperlink destination for PDF.
hyperlinkReport	string	Hyperlink destination for Report or page within Report.
justify	integer	Text justification for the item. Same rules as justify property of Item Class.
output	string	The output text from a Program or Library Program cell.
prefix	string	Prefix text to appear before Library Program output.
program	string	Path and filename for a Program cell, or the filename (e.g. *title.js*) for a Library Program cell.
programArgs	Array of strings	Program arguments
rightBorderWidth	real	Cell right border width. Can be 0.0, 0.1, 0.5, 0.75, 1.0, 1.5, 2.0, 2.5, 3.0, 4.0, 5.0. Other values will result in no border.
row	integer	The cell row index.
rowMergeOrigin	integer	The row index of the top-left cell in this merge cell group (if cell not merged then == row).
rowSpan	integer	Number of rows this cell spans (for merged cells). == 1 if not merged. Use rowMergeOrigin to find top-left cell.
suffix	string	Suffix text to appear after Library Program output.
text	string	The cell text. For Program and Library Program cells, use the **prefix**, **output** and **suffix** properties.
textColour	Colour object	Colour of text
topBorderWidth	real	Cell top border width. Can be 0.0, 0.1, 0.5, 0.75, 1.0, 1.5, 2.0, 2.5, 3.0, 4.0, 5.0. Other values will result in no border.
type	integer	Can be Item.TEXT,Item.LIBRARY_PROGRAM or Item.PROGRAM.

variable	string	REPORTER variable for library program output.
width	real	Cell width. Modifying this property will modify the width of all cells in the column.

Return type

object

Example

To get the properties of the top-left cell in a table:

```
i.GetCellProperties(0, 0);
```

GetColumnProperties(column*[integer]*, header*[constant]*)

Description

Get an autotable column properties. Valid for item type Item.AUTO_TABLE.

Arguments

- **column** (integer)

The index of the column of interest. Note that indices start from 0.

- **header** (constant)

An argument to signify to get the properties of the header or the generated rows. Can be Reporter.AUTO_TABLE_HEADER or Reporter.AUTO_TABLE_ROWS.

Returns

Object with the following properties:

Name	Type	Description
conditions	integer	Number of conditions assigned to this cell.
fillColour	Colour object	Fill colour
fontName	string	Font name (e.g. "Courier").
fontSize	integer	Font size (between 6 and 72).
fontStyle	integer	Font style. Same rules as fontStyle property of
hyperlinkHTML	string	Hyperlink destination for HTML.
hyperlinkPDF	string	Hyperlink destination for PDF.
hyperlinkReport	string	Hyperlink destination for Report or page within Report.
justify	integer	Text justification for the item. Same rules as justify property of Item Class.
program	string	Path and filename for a Program cell, or the filename (e.g. *title.js*) for a Library Program cell.
programArgs	Array of strings	Program arguments
text	string	The cell text. For Program and Library Program cells, use the **prefix**, **output** and **suffix** properties.
textColour	Colour object	Colour of text
type	integer	Can be Item.TEXT,Item.LIBRARY_PROGRAM or Item.PROGRAM.
width	real	Cell width. Modifying this property will modify the width of all cells in the column.

Return type

object

Example

Returns the column properties of the header of the first column:

```
i.GetColumnProperties(0, Reporter.AUTO_TABLE_HEADER);
```

GetColumnWidth(row*[integer]*)

Description

Get the width of a table column. Valid for item types Item.TABLE or Item.AUTO_TABLE.

Arguments

- **row** (integer)

The index of the column of interest. Note that indices start from 0.

Returns

Integer. The width of the specified column.

Return type

Number

Example

To get the width of the first column in a table:

```
i.GetColumnWidth(0);
```

GetCondition(index*[integer]*)

Description

Get the conditional formatting data for an item. Valid for item types Item.TEXT_FILE, Item.PROGRAM, Item.TEXT or Item.TEXTBOX (for Item.AUTO_TABLE and Item.TABLE, see GetCondition functions with additional arguments below).

Arguments

- **index** (integer)

The index of the condition to get. Note that indices start from 0. See conditions for the total number of comditions

Returns

Object with the following properties:

Name	Type	Description
fillColour	Colour object	Fill colour
fontName	string	Font name (e.g. "Courier").
fontSize	integer	Font size (between 6 and 72).
fontStyle	integer	Font style. See Text style constants for details.
justify	integer	Text alignment for the item. See Justification constants for details.
name	string	Condition name

textColour	Colour object	Colour of text
type	integer	See Condition types constants for details.
value	string	First condition value
value2	string	Second condition value (where relevant)

Return type

object

Example

To get the data for the 2nd condition in item i:

```
var condition = i.GetCondition(1);
```

GetCondition(index*[integer]*, column*[integer]*)

Description

Get the conditional formatting data for an Item.AUTO_TABLE item.

Arguments

- **index** (integer)

The index of the condition to get. Note that indices start from 0.

- **column** (integer)

The column to get the condition from. Note that indices start from 0.

Returns

Object with the following properties:

Name	Type	Description
fillColour	Colour object	Fill colour
fontName	string	Font name (e.g. "Courier").
fontSize	integer	Font size (between 6 and 72).
fontStyle	integer	Font style. See Text style constants for details.
justify	integer	Text alignment for the item. See Justification constants for details.
name	string	Condition name
textColour	Colour object	Colour of text
type	integer	See Condition types constants for details.
value	string	First condition value
value2	string	Second condition value (where relevant)

Return type

object

Example

To get the data for the 2nd condition from the 3rd column in autotable item i:

```
var condition = i.GetCondition(1, 2);
```

GetCondition(index*[integer]*, row*[integer]*, column*[integer]*)

Description

Get the conditional formatting data for an Item.TABLE item.

Arguments

- **index** (integer)

The index of the condition to get. Note that indices start from 0.

- **row** (integer)

The cell row to get the condition from. Note that indices start from 0.

- **column** (integer)

The cell column to get the condition from. Note that indices start from 0.

Returns

Object with the following properties:

Name	Type	Description
fillColour	Colour object	Fill colour
fontName	string	Font name (e.g. "Courier").
fontSize	integer	Font size (between 6 and 72).
fontStyle	integer	Font style. See Text style constants for details.
justify	integer	Text alignment for the item. See Justification constants for details.
name	string	Condition name
textColour	Colour object	Colour of text
type	integer	See Condition types constants for details.
value	string	First condition value
value2	string	Second condition value (where relevant)

Return type

object

Example

To get the data for the 2nd condition from the 4th row, 3rd column in table item i:

```
var condition = i.GetCondition(1, 3, 2);
```

GetFromName(page*[Page]*, name*[string]*) [static]

Description

Get an Item from its name.

Arguments

- **page** (Page)

Page to get item from

- **name** (string)

Item name

Returns

Item object (or null if item cannot be found)

Return type

Item

Example

To get the item with name test on page p:

```
var item = Item.GetFromName(p, "test");
```

GetGeneratedData(row_index*[integer]*, column_index*[integer]*)

Description

Get the text that appears in an autotable cell once generated. Valid for item type Item.AUTO_TABLE.

Arguments

- **row_index** (integer)

The index of the row of interest. Note that indices start from 0.

- **column_index** (integer)

The index of the column of interest. Note that indicies start from 0.

Returns

String: the text displayed in the specified row and column.

Return type

String

Example

Get the data from the first cell in the first row and column in an autotable.

```
i.GetGeneratedData(0, 0);
```

GetRowHeight(row*[integer]*)

Description

Get the height of a table row. Valid for item type Item.TABLE.

Arguments

- **row** (integer)

The index of the row of interest. Note that indices start from 0.

Returns

integer

Return type

Number

Example

To get the height of the first row in a table:

```
i.GetRowHeight(0);
```

InsertColumn(column*[integer]*)

Description

Insert a column into a table. Valid for item types Item.TABLE and Item.AUTO_TABLE.

Arguments

- **column** (integer)

The index of the position where the inserted column will end up. Note that indices start from 0. If no argument is given, a column will be added to the bottom of the table.

Returns

No return value

Example

To insert a column that will become the second column from the left of the table:

```
i.InsertColumn(1);
```

InsertRow(row*[integer]*)

Description

Insert a row into a table. Valid for item type Item.TABLE.

Arguments

- **row** (integer)

The index of the position where the inserted row will end up. Note that indices start from 0. If no argument is given, a row will be added to the bottom of the table.

Returns

No return value

Example

To insert a row that will become the second row from the top of the table:

```
i.InsertRow(1);
```

MergeCells(topLeftRow*[integer]*, topLeftColumn*[integer]*, rows*[integer]*, columns*[integer]*)

Description

Merge specified cells in a table. Valid for item types Item.TABLE and Item.AUTO_TABLE.

Arguments

- **topLeftRow** (integer)

The row index of the top-left cell in the group of cells to be merged. Note that indices start from 0.

- **topLeftColumn** (integer)

The column index of the top-left cell in the group of cells to be merged. Note that indices start from 0.

- **rows** (integer)

The number of rows of cells to be merged (measured from the topLeftRow position).

- **columns** (integer)

The number of columns of cells to be merged (measured from the topLeftColumn position).

Returns

No return value

Example

To merge the cells in first row and the first two columns in the table:

```
i.MergeCells(0, 0, 1, 2);
```

RemoveCondition(condition*[integer]*)

Description

Remove the specified condition for an item. Valid for item types Item.TEXT_FILE, Item.PROGRAM, Item.TEXT or Item.TEXTBOX (for Item.AUTO_TABLE and Item.TABLE, see RemoveCondition functions with additional arguments below).

Arguments

- **condition** (integer)

The index of the condition you wish to remove. Note that indices start at 0.

Returns

No return value

Example

To remove the condition for the condition index 1 in item i:

```
i.RemoveCondition(1);
```

RemoveCondition(condition*[integer]*, column*[integer]*)

Description

Remove the specified condition for an Item.AUTO_TABLE item.

Arguments

- **condition** (integer)

The index of the condition you wish to remove. Note that indices start at 0.

- **column** (integer)

The column to remove the condition for. Note that indices start from 0.

Returns

No return value

Example

To remove the condition for condition index 1 in the third column in item i:

```
i.RemoveCondition(1, 2);
```

RemoveCondition(condition*[integer]*, row*[integer]*, column*[integer]*)

Description

Remove the specified condition for an Item.TABLE item.

Arguments

- **condition** (integer)

The index of the condition you wish to remove. Note that indices start at 0.

- **row** (integer)

The row to remove the condition for. Note that indices start from 0.

- **column** (integer)

The column to remove the condition for. Note that indices start from 0.

Returns

No return value

Example

To remove the condition for condition index 1 in the fourth row, third column in item i:

```
i.RemoveCondition(1, 3, 2);
```

SetCellProperties(properties*[object]*, row*[integer]*, column*[integer]*)

Description

Set the properties of the specified cell. Valid for item type Item.TABLE.

Arguments

- **properties** (object)

An object containing the cell properties.

Object has the following properties:

Name	Type	Description
bottomBorderWidth (optional)	real	Cell bottom border width. Can be 0.0, 0.1, 0.5, 0.75, 1.0, 1.5, 2.0, 2.5, 3.0, 4.0, 5.0. Other values will result in no border.
fillColour (optional)	Colour object	Fill colour
fontName (optional)	string	Font name (e.g. "Courier").
fontSize (optional)	integer	Font size (between 6 and 72).
fontStyle (optional)	integer	Font style. See Text style constants for details.
hyperlinkHTML (optional)	string	Hyperlink destination for HTML.
hyperlinkPDF (optional)	string	Hyperlink destination for PDF.
hyperlinkReport (optional)	string	Hyperlink destination for Report or page within Report.
justify (optional)	integer	Text justification for the item. Same rules as justify property of Item Class.
prefix (optional)	string	Prefix text to appear before Library Program output.
program (optional)	string	Path and filename for a Program cell, or the filename (e.g. *title.js*) for a Library Program cell.

programArgs (optional)	Array of strings	Program arguments
rightBorderWidth (optional)	real	Cell right border width. Can be 0.0, 0.1, 0.5, 0.75, 1.0, 1.5, 2.0, 2.5, 3.0, 4.0, 5.0. Other values will result in no border.
suffix (optional)	string	Suffix text to appear after Library Program output.
text (optional)	string	The cell text. For Program and Library Program cells, use the **prefix**, **output** and **suffix** properties.
textColour (optional)	Colour object	Colour of text
topBorderWidth (optional)	real	Cell top border width. Can be 0.0, 0.1, 0.5, 0.75, 1.0, 1.5, 2.0, 2.5, 3.0, 4.0, 5.0. Other values will result in no border.
type (optional)	integer	Can be Item.TEXT,Item.LIBRARY_PROGRAM or Item.PROGRAM.
variable (optional)	string	REPORTER variable for library program output.

- **row** (integer)

The row index of the cell to be modified. Note that indices start from 0.

- **column** (integer)

The column index of the cell to be modified. Note that indices start from 0.

Returns

No return value

Example

To set the properties of the cell object to those of the object *cell_obj*:

```
i.SetCellProperties(cell_obj, 0, 0);
```

SetColumnProperties(properties*[object]*, column*[integer]*, header*[constant]*)

Description

Set the properties of an autotable column. Valid for item type Item.AUTO_TABLE.

Arguments

- **properties** (object)

Set the properties of an autotable column. Valid for item type Item.AUTO_TABLE.

Object has the following properties:

Name	Type	Description
fillColour (optional)	Colour object	Fill colour
fontName (optional)	string	Font name (e.g. "Courier").
fontSize (optional)	integer	Font size (between 6 and 72).
fontStyle (optional)	integer	Font style. Same rules as fontStyle property of
hyperlinkHTML (optional)	string	Hyperlink destination for HTML.
hyperlinkPDF (optional)	string	Hyperlink destination for PDF.
hyperlinkReport (optional)	string	Hyperlink destination for Report or page within Report.

justify (optional)	integer	Text justification for the item. Same rules as justify property of Item Class.
program (optional)	string	Path and filename for a Program cell, or the filename (e.g. *title.js*) for a Library Program cell.
programArgs (optional)	Array of strings	Program arguments
text (optional)	string	The cell text. For Program and Library Program cells, use the **prefix**, **output** and **suffix** properties.
textColour (optional)	Colour object	Colour of text
type (optional)	integer	Can be Item.TEXT,Item.LIBRARY_PROGRAM or Item.PROGRAM.

- **column** (integer)

The index of the column of interest. Note that indices start from 0.

- **header** (constant)

An argument to signify to set the properties of the header or the generated rows. Can be Reporter.AUTO_TABLE_HEADER or Reporter.AUTO_TABLE_ROWS.

Returns

No return value

Example

Sets the column properties of the header of the first column with the properties of the object *column_obj*.

```
i.SetColumnProperties(column_obj, 0, Reporter.AUTO_TABLE_HEADER);
```

SetColumnWidth(column*[integer]*, width*[real]*)

Description

Set the width of a table column. Valid for item type Item.TABLE.

Arguments

- **column** (integer)

The index of the column of interest. Note that indices start from 0.

- **width** (real)

The column width.

Returns

No return value

Example

To set the width of the first column in a table to 10.0:

```
i.SetColumnWidth(0, 10.0);
```

SetCondition(condition*[integer]*, properties*[object]*)

Description

Set the specified condition for an item. Valid for item types Item.TEXT_FILE, Item.PROGRAM, Item.TEXT or Item.TEXTBOX (for Item.AUTO_TABLE and Item.TABLE, see SetCondition functions with additional arguments below).

Arguments

- **condition** (integer)

The index of the condition you wish to set. Note that indices start at 0. If a condition already exists at the specified index, it will be replaced. To add a new condition, specify an index equal to the number of existing conditions.

- **properties** (object)

The index of the condition you wish to set. Note that indices start at 0. If a condition already exists at the specified index, it will be replaced. To add a new condition, specify an index equal to the number of existing conditions.

Object has the following properties:

Name	Type	Description
fillColour (optional)	Colour object	Fill colour
fontName (optional)	string	Font name (e.g. "Courier").
fontSize (optional)	integer	Font size (between 6 and 72).
fontStyle (optional)	integer	Font style. See Text style constants for details.
justify (optional)	integer	Text alignment for the item. See Justification constants for details.
name	string	Condition name
textColour (optional)	Colour object	Colour of text
type (optional)	integer	See Condition types constants for details.
value (optional)	string	First condition value
value2 (optional)	string	Second condition value (where relevant)

Returns

No return value

Example

To set the conditions for the condition index 1 in item i to those of the object obj:

```
var obj = { name:"example", type:Reporter.CONDITION_EQUAL_TO, value:"Test",
textColour:Colour.Red() };
i.SetCondition(1, obj);
```

SetCondition(condition*[integer]*, column*[integer]*, properties*[object]*)

Description

Set the specified condition for an Item.AUTO_TABLE item.

Arguments

- **condition** (integer)

The index of the condition you wish to set. Note that indices start at 0. If a condition already exists at the specified index, it will be replaced. To add a new condition, specify an index equal to the number of existing conditions.

- **column** (integer)

The column to set the condition for. Note that indices start from 0.

- **properties** (object)

The column to set the condition for. Note that indices start from 0.

Object has the following properties:

Name	Type	Description
fillColour (optional)	Colour object	Fill colour

fontName (optional)	string	Font name (e.g. "Courier").
fontSize (optional)	integer	Font size (between 6 and 72).
fontStyle (optional)	integer	Font style. See Text style constants for details.
justify (optional)	integer	Text alignment for the item. See Justification constants for details.
name	string	Condition name
textColour (optional)	Colour object	Colour of text
type (optional)	integer	See Condition types constants for details.
value (optional)	string	First condition value
value2 (optional)	string	Second condition value (where relevant)

Returns

No return value

Example

To set the conditions for condition index 1 in the third column in item i to those of the object obj:

```
var obj = { name:"example", type:Reporter.CONDITION_EQUAL_TO, value:"Test",
textColour:Colour.Red() };
i.SetCondition(1, 2, obj);
```

SetCondition(condition*[integer]*, row*[integer]*, column*[integer]*, properties*[object]*)

Description

Set the specified condition for an Item.TABLE item.

Arguments

- **condition** (integer)

The index of the condition you wish to set. Note that indices start at 0. If a condition already exists at the specified index, it will be replaced. To add a new condition, specify an index equal to the number of existing conditions.

- **row** (integer)

The row to set the condition for. Note that indices start from 0.

- **column** (integer)

The column to set the condition for. Note that indices start from 0.

- **properties** (object)

The column to set the condition for. Note that indices start from 0.

Object has the following properties:

Name	Type	Description
fillColour (optional)	Colour object	Fill colour
fontName (optional)	string	Font name (e.g. "Courier").
fontSize (optional)	integer	Font size (between 6 and 72).
fontStyle (optional)	integer	Font style. See Text style constants for details.
justify (optional)	integer	Text alignment for the item. See Justification constants for details.
name	string	Condition name
textColour (optional)	Colour object	Colour of text

type (optional)	integer	See Condition types constants for details.
value (optional)	string	First condition value
value2 (optional)	string	Second condition value (where relevant)

Returns

No return value

Example

To set the conditions for condition index 1 in the fourth row, third column in item i to those of the object obj:

```
var obj = { name:"example", type:Reporter.CONDITION_EQUAL_TO, value:"Test",
textColour:Colour.Red()
}; i.SetCondition(1, 3, 2, obj);
```

SetRowHeight(row*[integer]*, height*[real]*)

Description

Set the height of a table row. Valid for item type Item.TABLE and Item.AUTO_TABLE.

Arguments

- **row** (integer)

The index of the row of interest. Note that indices start from 0.

- **height** (real)

The row height.

Returns

No return value

Example

To set the height of the first row in a table to 10.0:

```
i.SetRowHeight(0, 10.0);
```

UnmergeCells(row*[integer]*, column*[integer]*)

Description

Unmerge the specified cell in a table. All cells merged to the specified cell will be unmerged. Valid for item types Item.TABLE and Item.AUTO_TABLE.

Arguments

- **row** (integer)

The row index of the cell to be unmerged. Note that indices start from 0.

- **column** (integer)

The column index of the cell to be unmerged. Note that indices start from 0..

Returns

No return value

Example

To unmerge the top-left cell in a table:

```
i.UnmergeCells(0, 0);
```

Options class

The Options class enables you to access several options in REPORTER. More...

The REPORTER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Options constants

Constants for Promises

Name	Description
Options.RUN_PROMISE_CONSTRUCTOR	Allow/run promises when an API constructor is called
Options.RUN_PROMISE_METHOD	Allow/run promises when an API method is called
Options.RUN_PROMISE_PROPERTY	Allow/run promises when an API property getter/setter is done
Options.RUN_PROMISE_SCRIPT	Allow/run promises when a script is run
Options.RUN_PROMISE_WINDOW_LOOP	Allow/run promises in a window event loop

Options class properties

Name	Type	Description
run_ promises	constant	When any promise callbacks/handlers are allowed to run. Can be a bitwise OR of: Options.RUN_ PROMISE_WINDOW_LOOP, Options.RUN_PROMISE_CONSTRUCTOR, Options.RUN_ PROMISE_METHOD and Options.RUN_PROMISE_PROPERTY Options.RUN_PROMISE_ SCRIPT The default is for all to be allowed. Promise handlers can also be run manually by using Utils.CallPromiseHandlers()

Detailed Description

The Options class is used to get/set options that REPORTER uses for certain functions. The options are available as **class** properties. See the documentation for more details. An example: Options.run_promises = Options.RUN_ PROMISE_WINDOW_LOOP

Page class

The Page class gives access to pages in Reporter. More...

The REPORTER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Member functions

- DeleteItem(index*[integer]*)
- Duplicate(index (optional)*[integer]*)
- Generate()
- GetAllItems()
- GetItem(index*[integer]*)
- ImportItem(filename*[string]*)

Page properties

Name	Type	Description
generating (read only)	logical	true if the entire page is currently being generated
items (read only)	integer	The total number of items on the page
master (read only)	logical	true if this is a master page object.
name	string	Name of the Page

Detailed Description

The Page class allows you to access the pages in templates that Reporter currently has open.

Constructor

new Page(template*[Template]*, options (optional)*[object]*)

Description

Create a new Page..

Arguments

- **template** (Template)

Template to create page in

- **options (optional)** (object)

Options specifying various page properties, including where the page should be created. If omitted, the default values below will be used.

Object has the following properties:

Name	Type	Description
colour (optional)	Colour object	Page background colour (white if omitted)

index (optional)	integer	The page index at which the new page will be inserted (indices start from zero). You cannot create pages prior to the current page i.e. the index must be greater than the index of the current page. If omitted, the new page will be created immediately after the current page. Note that the current page continues to be the page that the Script item is running on (it does not change to the newly-created page).
name (optional)	string	Name for page (empty if omitted)

Returns

Page object

Return type

Page

Example

To create a new blank Page object in template *t*:

```
var page = new Page(t);
```

To create a new red page named "Last page" as the last page in template *t*:

```
var page = new Page(t, {name:"Last page", colour:Colour.Red(),
index:t.GetAllPages().length});
```

Details of functions

DeleteItem(index*[integer]*)

Description

Deletes an item from a page.

Arguments

- **index** (integer)

The index of the item that you want to delete. Note that indices start at 0.

Returns

No return value

Example

To delete the first item of page *p*:

```
p.DeleteItem(0);
```

Duplicate(index (optional)*[integer]*)

Description

Duplicate a page

Arguments

- **index (optional)** (integer)

The page index that you want to insert the duplicate page at in the template. Note that indices start at 0. If omitted the duplicate page will be put after the one that you are duplicating.

Returns

Page object

Return type

Page

Example

To duplicate page p:

```
var dp = p.Duplicate();
```

To duplicate page p putting the duplicate as the first page in the template:

```
var dp = p.Duplicate(0);
```

Generate()

Description

Generate a page

Arguments

No arguments

Returns

no return value

Example

To generate page p:

```
p.Generate();
```

GetAllItems()

Description

Gets all of the items from a page.

Arguments

No arguments

Returns

Array of Item objects

Return type

Array

Example

To get all of the items on page p:

```
var items = p.GetAllItems();
```

GetItem(index*[integer]*)

Description

Get an item from a page.

Arguments

- **index** (integer)

The index of the item on the page that you want to get. Note that indices start at 0.

Returns

Item

Return type

Item

Example

To get the 1st item on page p:

```
p.GetItem(0);
```

ImportItem(filename*[string]*)

Description

Import an item from a file onto the page.

Arguments

- **filename** (string)

File containing the object to import

Returns

Item

Return type

Item

Example

To read an item from file "item.oro" and put it on page p:

```
p.ImportItem("item.oro");
```

Reporter class

The Reporter class contains constants for use in REPORTER. More...

The REPORTER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Reporter constants

Constants for Autotable source and row types

Name	Description
Reporter.AUTO_TABLE_DIRECTORY	Autotable data is generated from a directory.
Reporter.AUTO_TABLE_FILE	Autotable data is generated from a file.
Reporter.AUTO_TABLE_HEADER	Represents the header row in an Autotable.
Reporter.AUTO_TABLE_ROWS	Represents the rows with generated data in an Autotable.

Constants for Condition types

Name	Description
Reporter.CONDITION_BETWEEN	Treats the value as a number. True if value is between the two condition values
Reporter.CONDITION_CONTAINS_STRING	Treats the vlue as a string. True if the value contains the string
Reporter.CONDITION_DOESNT_CONTAIN_STRING	Treats the vlue as a string. True if the value does not contain the string
Reporter.CONDITION_DOESNT_MATCH_REGEX	Treats the value as a regular expression. True if the regular expression does not match
Reporter.CONDITION_EQUAL_TO	Treats the value as a string. True if the strings are equal
Reporter.CONDITION_GREATER_THAN	Treats the value as a number. True if value is greater than the condition value
Reporter.CONDITION_LESS_THAN	Treats the value as a number. True if value is less than the condition value
Reporter.CONDITION_MATCHES_REGEX	Treats the value as a regular expression. True if the regular expression matches
Reporter.CONDITION_NOT_BETWEEN	Treats the value as a number. True if value is between the two condition values
Reporter.CONDITION_NOT_EQUAL_TO	Treats the value as a string. True if the strings are not equal

Constants for Justification

Name	Description
Reporter.JUSTIFY_BOTTOM	Bottom justification of text

Reporter.JUSTIFY_CENTRE	Centre justification of text
Reporter.JUSTIFY_LEFT	Left justification of text
Reporter.JUSTIFY_MIDDLE	Middle justification of text
Reporter.JUSTIFY_RIGHT	Right justification of text
Reporter.JUSTIFY_TOP	Top justification of text

Constants for Line cap style

Name	Description
Reporter.CAP_FLAT	A square line ending at the end point of the line
Reporter.CAP_ROUND	A rounded line ending
Reporter.CAP_SQUARE	A square line that extends beyond the end point of the line by half the line width

Constants for Line join style

Name	Description
Reporter.JOIN_BEVEL	The triangular notch where the line segments meet is filled
Reporter.JOIN_MITRE	The outer edges of the line segments are extended to meet at an angle and this is filled
Reporter.JOIN_ROUND	A circular arc between the two line segments is filled

Constants for Line style

Name	Description
Reporter.LINE_DASH	A dashed line (dashes separated by a few pixels)
Reporter.LINE_DASH_DOT	A line drawn with alternate dashes and dots
Reporter.LINE_DASH_DOT_DOT	A line drawn with one dash and two dots
Reporter.LINE_DOT	A dotted line (dots separated by a few pixels)
Reporter.LINE_NONE	Invisible line
Reporter.LINE_SOLID	A simple continuous line

Constants for Text style

Name	Description
Reporter.TEXT_BOLD	Text drawn in a bold font
Reporter.TEXT_ITALIC	Text drawn in an italic font
Reporter.TEXT_NORMAL	Text drawn in a normal font
Reporter.TEXT_UNDERLINE	Text drawn underlined

Constants for View

Name	Description

Reporter.VIEW_DESIGN	Show template in design view
Reporter.VIEW_PRESENTATION	Show template in presentation view

Reporter properties

Detailed Description

The Reporter class allows you to access constants used in REPORTER.

Template class

The Template class gives access to templates in Reporter. More...

The REPORTER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- GetAll()
- GetCurrent()

Member functions

- Close()
- DeletePage(index*[integer]*)
- DeleteTemporaryVariables()
- DownloadThumbnail(filepath*[string]*)
- EditVariables(title (optional)*[string]*, message (optional)*[string]*, update (optional)*[boolean]*, variables (optional)*[array]*, columns (optional)*[constant]*, alphabetical (optional)*[boolean]*)
- ExpandVariablesInString(string*[string]*)
- Generate()
- GetAllPages()
- GetMaster()
- GetPage(index*[integer]*)
- GetVariableDescription(name*[string]*)
- GetVariableValue(name*[string]*)
- Html(filename*[string]*)
- Pdf(filename*[string]*)
- Pptx(filename*[string]*)
- Print(printer*[string]*)
- Save()
- SaveAs(filename*[string]*, readonly (optional)*[boolean]*)
- Update()
- UpdateThumbnail(filepath*[string]*)

Template properties

Name	Type	Description
filename (read only)	string	Filename (without path) of the Template.
generating (read only)	logical	true if the entire template is currently being generated
pages (read only)	integer	Number of Pages in template
path (read only)	string	Absolute path (without filename) of the Template. If the Template is new and has not yet been saved, this property will be empty.
readonly (read only)	logical	true if the template is read-only
view	constant	Current view type (presentation or design view) for this Template. Can be: Reporter.VIEW_DESIGN or Reporter.VIEW_PRESENTATION.

Detailed Description

The Template class allows you to access the templates that Reporter currently has open.
Note that if you want to get a list of the current templates in Reporter you should see the templates array in the reporter object.
The currently active template is stored in the currentTemplate property of the reporter object.

Constructor

new Template(filename (optional)*[string]*)

Description

Create a new Template. The filename argument is optional. If present it is a file to open

Arguments

- **filename (optional)** (string)

Name of template file to open

Returns

Template object

Return type

Template

Example

To create a new blank Template object

```
var template = new Template();
```

Details of functions

Close()

Description

Close a template.
Note that if you call this function for a Template object, the Template data will be deleted, so you should not try to use it afterwards!.

Arguments

No arguments

Returns

no return value

Example

To close template data:

```
data.Close();
```

DeletePage(index*[integer]*)

Description

Deletes a page from a template.

Arguments

- **index** (integer)

The index of the page that you want to delete. Note that indices start at 0.

Returns

No return value

Example

To delete the first page of template t:

```
t.DeletePage(0);
```

DeleteTemporaryVariables()

Description

Deletes any temporary variables from a template.

Arguments

No arguments

Returns

No return value

Example

To delete all the temporary variables from template t:

```
t.DeleteTemporaryVariables();
```

DownloadThumbnail(filepath*[string]*)

Description

Downloads the thumbnail of the template

Arguments

- **filepath** (string)

File path for the thumbnail.

Returns

Boolean indicating success or failure.

Return type

Boolean

Example

```
var success =
template.DownloadThumbnail("/home/username/project1/template1.png");
```

EditVariables(title (optional)*[string]*, message (optional)*[string]*, update (optional)*[boolean]*, variables (optional)*[array]*, columns (optional)*[constant]*, alphabetical (optional)*[boolean]*)

Description

Start a dialog to edit the template variables

Arguments

- **title (optional)** (string)

Title for dialog. If omitted, null or an empty string is given then the default title will be used.

- **message (optional)** (string)

Message to show in dialog. If omitted, null or an empty string is given then the default message will be used.

- **update (optional)** (boolean)

Whether the variables in the template will be updated with the new values if OK is pressed. Setting this to be false allows you to check variable values before updating them from a script. If omitted the default is true

- **variables (optional)** (array)

A list of variables to show in the dialog. If omitted, null or an empty array, all variables will be shown

- **columns (optional)** (constant)

Columns to show in the dialog (as well as the variable value column). Can be a bitwise OR of Variable.NAME, Variable.TYPE, Variable.DESCRIPTION, Variable.FORMAT, Variable.PRECISION and Variable.TEMPORARY. If omitted columns will be shown for name and description

- **alphabetical (optional)** (boolean)

Whether to sort variables in the table by alphabetical order. If false, variables are listed in the order they are passed in the optional variables argument. If no variables are passed to the function, all template variables will be shown in alphabetical order. If omitted, the default value is true.

Returns

Object containing the variable names and values or null if cancel was pressed.

Return type

Object

Example

To edit all of the variables in template:

```
var variables = template.EditVariables();
```

To edit variables TEST and EXAMPLE in template giving a title and a message, returning the edited values but **not** updating them in the template:

```
var variables = template.EditVariables("Edit variables",
"Type in the values", false, ["TEST", "EXAMPLE");
```

ExpandVariablesInString(string*[string]*)

Description

Replaces any variables in a string with their current values

Arguments

- **string** (string)

The string you want to expand variables in.

Returns

String (string) with variables expanded. If a variable in a string does not exist it is replaced by a blank.

Return type

String

Example

If the variable FRED in template contains the value "test", then the following

```
var value
= template.ExpandVariablesInString("This is a %FRED%");
```

will return "This is a test" in variable value.

Generate()

Description

Generate a template

Arguments

No arguments

Returns

no return value

Example

To generate template data:

```
data.Generate();
```

GetAll() [static]

Description

Get all of the open templates

Arguments

No arguments

Returns

array of Template objects or null if no open templates

Return type

Array

Example

To get all of the templates open in REPORTER:

```
var templates = Template.GetAll();
```

GetAllPages()

Description

Gets all of the pages from a template.

Arguments

No arguments

Returns

Array of Page objects

Return type

Array

Example

To get all of the pages from template t:

```
var pages = t.GetAllPages();
```

GetCurrent() [static]

Description

Get the currently active template

Arguments

No arguments

Returns

Template object or null if no active template

Return type

Template

Example

To get the current template open in REPORTER:

```
var current_template = Template.GetCurrent();
```

GetMaster()

Description

Get the master page from a template.

Arguments

No arguments

Returns

Page object

Return type

Page

Example

To get the master page of template t:

```
var m = t.GetMaster();
```

GetPage(index*[integer]*)

Description

Get a page from a template.

Arguments

- **index** (integer)

The index of the page that you want to get. Note that indices start at 0.

Returns

Page object

Return type

Page

Example

To get the first page of template t:

```
var p = t.GetPage(0);
```

GetVariableDescription(name*[string]*)

Description

Get the description for a variable

Arguments

- **name** (string)

Variable name you want to get description for.

Returns

Variable description (string) or null if variable does not exist

Return type

String

Example

To get description for variable FRED in template:

```
var description = template.GetVariableDescription("FRED");
```

GetVariableValue(name*[string]*)

Description

Get the value for a variable

Arguments

- **name** (string)

Variable name you want to get value for.

Returns

Variable value (string) or null if variable does not exist

Return type

String

Example

To get value for variable FRED in template:

```
var value = template.GetVariableValue("FRED");
```

Html(filename*[string]*)

Description

Save a template as HTML

Arguments

- **filename** (string)

Filename you want to save.

Returns

no return value

Example

To save template data as file /data/test/template.html:

```
data.Html("/data/test/template.html");
```

Pdf(filename*[string]*)

Description

Save a template as Adobe Acrobat PDF

Arguments

- **filename** (string)

Filename you want to save.

Returns

no return value

Example

To save template data as file /data/test/template.pdf:

```
data.Pdf("/data/test/template.pdf");
```

Pptx(filename*[string]*)

Description

Save a template as PowerPoint

Arguments

- **filename** (string)

Filename you want to save.

Returns

no return value

Example

To save template data as file /data/test/template.pptx:

```
data.Pptx("/data/test/template.pptx");
```

Print(printer*[string]*)

Description

Print template on a printer

Arguments

- **printer** (string)

Printer you want to print to.

Returns

no return value

Example

To print template data on printer myprinter:

```
data.Print("myprinter");
```

Save()

Description

Save a template

Arguments

No arguments

Returns

no return value

Example

To save template data:

```
data.Save();
```

SaveAs(filename*[string]*, readonly (optional)*[boolean]*)

Description

Save a template/report with a new name

Arguments

- **filename** (string)

Filename you want to save. Note if you use the .orr extension the template will be saved as a report if generated.

- **readonly (optional)** (boolean)

If saved template/report will be readonly or not.

Returns

no return value

Example

To save template data as file /data/test/template.opt:

```
data.SaveAs("/data/test/template.opt");
```

Update()

Description

Update/redraw a template

Arguments

No arguments

Returns

no return value

Example

To update template data:

```
data.Update();
```

UpdateThumbnail(filepath*[string]*)

Description

Updates the thumbnail of the template

Arguments

- **filepath** (string)

File path for the thumbnail.

Returns

Boolean indicating success or failure.

Return type

Boolean

Example

```
var success = template.UpdateThumbnail("/home/username/project1/template1.png");
```

Utils class

The Utils class contains various useful utility functions. More...

The REPORTER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Ascii85Decode(encoded*[string]*)
- Ascii85Encode(data*[ArrayBuffer]*, length (optional)*[integer]*)
- Build()
- CallPromiseHandlers()
- GarbageCollect()
- HiResTimer()
- SHA256(filename*[string]*)
- SHA512(filename*[string]*)
- Terminate()
- TimerResolution()
- UUID()
- Version()

Detailed Description

The Utils class is used to provide various useful functions.

Details of functions

Ascii85Decode(encoded*[string]*) [static]

Description

Decodes an ASCII85 encoded string. See Utils.Ascii85Encode() for details on the method.

Arguments

- **encoded** (string)

An ASCII85 encoded string

Returns

ArrayBuffer object

Return type

ArrayBuffer

Example

To decode an ASCII85 encoded string:

```
var decoded = Utils.Ascii85Decode(encoded);
```

Ascii85Encode(data*[ArrayBuffer]*, length (optional)*[integer]*) [static]

Description

Encodes an ASCII85 encoded string. This enables binary data to be represented by ASCII characters using five ASCII characters to represent four bytes of binary data (making the encoded size 1/4 larger than the original). By doing this binary data can be stored in JavaScript strings. Note that the method used by PRIMER to encode and decode strings differs from the standard ASCII85 encoding as that uses the ASCII characters ", ’ and \ which cannot be used in JavaScript strings as they have special meanings. The method in PRIMER uses
0-84 are !-u (ASCII codes 33-117) (i.e. 33 is added to it) with the following exceptions
v is used instead of " (ASCII code 118 instead of 34)
w is used instead of ’ (ASCII code 119 instead of 39)
x is used instead of \ (ASCII code 120 instead of 92)
If all five digits are 0 they are represented by a single character z instead of !!!!!

Arguments

- **data** (ArrayBuffer)

ArrayBuffer containing the data

- **length (optional)** (integer)

Length of data in array buffer to encode. If omitted the whole array buffer will be encoded

Returns

string

Return type

String

Example

To encode ArrayBuffer data:

```
var encoded = Utils.Ascii85Encode(data);
```

Build() [static]

Description

Returns the build number

Arguments

No arguments

Returns

integer

Return type

Number

Example

To get the current build number

```
var build = Utils.Build();
```

CallPromiseHandlers() [static]

Description

Manually call any promise handlers/callbacks in the job queue

Arguments

No arguments

Returns

no return value

Example

To run any queued promise handlers/callbacks:

```
Utils.CallPromiseHandlers();
```

GarbageCollect() [static]

Description

Forces garbage collection to be done. This should not normally need to be called but in exceptional circumstances it can be called to ensure that garbage collection is done to return memory.

Arguments

No arguments

Returns

no return value

Example

To force garbage collection to be done:

```
Utils.GarbageCollect();
```

HiResTimer() [static]

Description

A high resolution timer that can be used to time how long things take. The first time this is called the timer will start and return 0. Subsequent calls will return the time in nanoseconds since the first call. Note that the timer will almost certainly not have 1 nanosecond precision but, depending on the platform, should should have a resolution of at least 1 microsecond. The resolution can be found by using Utils.TimerResolution()

Arguments

No arguments

Returns

number

Return type

number

Example

To time how long something takes to nanosecond precision:

```
var start = Utils.HiResTimer();
do something that takes some time...
var end = Utils.HiResTimer();
Message("it took " + (end-start) + "nanoseconds");
```

SHA256(filename*[string]*) [static]

Description

Create a SHA-256 hash for a file

Arguments

- **filename** (string)

File to calculate the hash for

Returns

string

Return type

String

Example

To generate the hash for file "example.txt":

```
var sha256 = Utils.SHA256("example.txt");
```

SHA512(filename*[string]*) [static]

Description

Create a SHA-512 hash for a file

Arguments

- **filename** (string)

File to calculate the hash for

Returns

string

Return type

String

Example

To generate the hash for file "example.txt":

```
var sha512 = Utils.SHA512("example.txt");
```

Terminate() [static]

Description

Terminates the program. Note that this **terminates the entire program**, not the script. To terminate the script use Exit() instead

Arguments

No arguments

Returns

No return value

Example

To terminate the program:

```
Utils.Terminate();
```

TimerResolution() [static]

Description

Returns the resolution (precision) of the Utils.HiResTimer() timer in nanoseconds

Arguments

No arguments

Returns

number

Return type

number

Example

To find the resolution of the timer in nanoseconds:

```
var resolution = Utils.TimerResolution();
```

UUID() [static]

Description

Create an UUID (Universally Unique Identifier)

Arguments

No arguments

Returns

string

Return type

String

Example

To generate an UUID:

```
var uuid = Utils.UUID();
```

Version() [static]

Description

Returns the version number

Arguments

No arguments

Returns

real

Return type

Number

Example

To get the current version number

```
var version = Utils.Version();
```

Variable class

The Variable class gives access to variables in Reporter. More...

The REPORTER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- GetAll(template*[Template]*)
- GetFromName(template*[Template]*, name*[string]*)

Member functions

- Remove()

Variable constants

Name	Description
Variable.DESCRIPTION	Show variable description when editing variables with Template.EditVariables()
Variable.FORMAT	Show variable format when editing variables with Template.EditVariables()
Variable.FORMAT_FLOAT	Variable has floating point number format
Variable.FORMAT_GENERAL	Variable has general format
Variable.FORMAT_INTEGER	Variable has integer format
Variable.FORMAT_ LOWERCASE	Variable has lower case format
Variable.FORMAT_NONE	Variable has no format
Variable.FORMAT_SCIENTIFIC	Variable has scientific format
Variable.FORMAT_UPPERCASE	Variable has upper case format
Variable.NAME	Show variable name when editing variables with Template.EditVariables()
Variable.PRECISION	Show variable precision when editing variables with Template.EditVariables()
Variable.READONLY	Show variable readonly status when editing variables with Template.EditVariables()
Variable.TEMPORARY	Show variable temporary status when editing variables with Template.EditVariables()
Variable.TYPE	Show variable type when editing variables with Template.EditVariables()
Variable.VALUE	Show variable value when editing variables with Template.EditVariables()

Variable properties

Name	Type	Description
description	string	Variable description

format	constant	Variable format. Can be Variable.FORMAT_NONE, Variable.FORMAT_FLOAT, Variable.FORMAT_SCIENTIFIC, Variable.FORMAT_GENERAL, Variable.FORMAT_INTEGER, Variable.FORMAT_UPPERCASE or Variable.FORMAT_LOWERCASE
name	string	Variable name
precision	integer	Variable precision for floating point numbers.
readonly	logical	If Variable is read only or not.
temporary	logical	If Variable is temporary or not.
type	string	Variable type. Predefined types are "Directory", "File(absolute)", "File(basename)", "File(extension)", "File(tail)", "General", "Number" and "String". Alternatively give your own type. e.g. "NODE ID"
value	string	Variable value

Detailed Description

The Variable class allows you to access the name, description and value of a variable inside Reporter.
Note that if you want to get a list of the variables used in a Template you should see the variables array in the Template object.
The name, description and value properties give access to the variable name, description and value respectively.

Constructor

new Variable(template*[Template]*, name*[string]*, description (optional)*[string]*, value (optional)*[string]*, type (optional)*[string]*, readonly (optional)*[boolean]*, temporary (optional)*[boolean]*)

Description

Create a new Variable. The template and name arguments MUST be given, all others are optional

Arguments

- **template** (Template)

Template object to create variable in

- **name** (string)

Name of variable

- **description (optional)** (string)

Description of variable

- **value (optional)** (string)

Variable value

- **type (optional)** (string)

Type of variable. Predefined types are "Directory", "File(absolute)", "File(basename)", "File(extension)", "File(tail)", "General", "Number" and "String". Alternatively give your own type. e.g. "NODE ID". If omitted default is "General"

- **readonly (optional)** (boolean)

If variable is readonly or not. If omitted default is false.

- **temporary (optional)** (boolean)

If variable is temporary or not. If omitted default is true.

Returns

Variable object

Return type

Variable

Example

To create a new Variable object called TEST with description 'test variable', type of "Number" and value '10' which is not readonly for template, templ

```
var variable = new Variable(templ, "TEST", "test variable", "10", "Number",
false);
```

Details of functions

GetAll(template*[Template]*) [static]

Description

Returns an array of Variable objects for all of the variables in a Template.

Arguments

- **template** (Template)

Template to get the variables from

Returns

Array of Variable objects

Return type

Array

Example

To get all the variables in template t:

```
var v = Variable.GetAll(t);
```

GetFromName(template*[Template]*, name*[string]*) [static]

Description

Returns the Variable object for a variable name.

Arguments

- **template** (Template)

Template to find the variable in

- **name** (string)

name of the variable you want the Variable object for

Returns

Variable object (or null if variable does not exist)

Return type

Variable

Example

To get the Variable object for variable EXAMPLE in template t:

```
var v = Variable.GetFromName(t, "EXAMPLE");
```

Remove()

Description

Remove a variable
Note that if you call this function for a Variable object, the Variable data will be deleted, so you should not try to use it afterwards!.

Arguments

No arguments

Returns

no return value

Example

To remove variable data:

```
data.Remove();
```

Window class

The Window class gives access to windows for a graphical user interface. More...

The REPORTER JavaScript API provides many class constants, properties and methods. For Arup to be able to extend and enhance the API in the future any constant, property or method names beginning with a lowercase or uppercase letter are reserved.
If you need to add your own properties or methods to one of the existing classes then to avoid any potential future conflict you should ensure that the name begins with either an underscore (_) or a dollar sign ($) or the name is prefixed with your own unique identifier.
For example if company 'ABC' need to add a property called 'example' then to avoid any potential future conflict use one of:

- _example
- $example
- ABC_example

Class functions

- Error(title*[string]*, error*[string]*, buttons (optional)*[constant]*)
- GetDirectory(initial (optional)*[string]*)
- GetFile(extension (optional)*[string]*, allow new (optional)*[boolean]*, initial (optional)*[string]*)
- GetFiles(extension (optional)*[string]*)
- GetInteger(title*[string]*, message*[string]*)
- GetNumber(title*[string]*, message*[string]*)
- GetOptions(title*[string]*, message*[string]*, options*[object]*)
- GetString(title*[string]*, message*[string]*)
- Information(title*[string]*, info*[string]*, buttons (optional)*[constant]*)
- Message(title*[string]*, message*[string]*, buttons (optional)*[constant]*)
- Question(title*[string]*, question*[string]*, buttons (optional)*[constant]*)
- Warning(title*[string]*, warning*[string]*, buttons (optional)*[constant]*)

Window constants

Name	Description
Window.CANCEL	Show CANCEL button
Window.NO	Show NO button
Window.OK	Show OK button
Window.YES	Show YES button

Detailed Description

The Window class is used to define several standard windows that can be used to read data, give messages and provide feedback

Details of functions

Error(title*[string]*, error*[string]*, buttons (optional)*[constant]*) [static]

Description

Show an error message in a window.

Arguments

- **title** (string)

Title for window.

- **error** (string)

Error message to show in window.

- **buttons (optional)** (constant)

The buttons to use. Can be bitwise OR of Window.OK, Window.CANCEL, Window.YES or Window.NO. If this is

omitted an OK button will be used.

Returns

Button pressed

Return type

Number

Example

To show error *Critical error!\nAbort?* in window with title *Error* with Yes and No buttons:

```
var answer = Window.Error("Error", "Critical error!\nAbort?", Window.YES |
Window.NO);
if (answer == Window.YES) Exit();
```

GetDirectory(initial (optional)*[string]*) [static]

Description

Map the directory selector box native to your machine, allowing you to choose a directory.

Arguments

- **initial (optional)** (string)

Initial directory to start from.

Returns

directory (string), (or null if cancel pressed).

Return type

String

Example

To select a directory:

```
var dir = Window.GetDirectory();
```

GetFile(extension (optional)*[string]*, allow new (optional)*[boolean]*, initial (optional)*[string]*) [static]

Description

Map a file selector box allowing you to choose a file. See also Window.GetFiles()

Arguments

- **extension (optional)** (string)

Extension to filter by.

- **allow new (optional)** (boolean)

Allow creation of new file.

- **initial (optional)** (string)

Initial directory to start from.

Returns

filename (string), (or null if cancel pressed).

Return type

String

Example

To select a file using extension '.key':

```
var file = Window.GetFile(".key");
```

GetFiles(extension (optional)*[string]*) [static]

Description

Map a file selector box allowing you to choose multiple files. See also Window.GetFile()

Arguments

- **extension (optional)** (string)

Extension to filter by.

Returns

Array of filenames (strings), or null if cancel pressed.

Return type

String

Example

To select multiple files using extension '.key':

```
var files = Window.GetFiles(".key");
```

GetInteger(title*[string]*, message*[string]*) [static]

Description

Map a window allowing you to input an integer. OK and Cancel buttons are shown.

Arguments

- **title** (string)

Title for window.

- **message** (string)

Message to show in window.

Returns

Integer. Value input, (or null if cancel pressed).

Return type

Number

Example

To create an input window with title *Input* and message *Input integer* and return the value input:

```
var value = Window.GetInteger("Input", "Input integer");
```

GetNumber(title*[string]*, message*[string]*) [static]

Description

Map a window allowing you to input a number. OK and Cancel buttons are shown.

Arguments

- **title** (string)

Title for window.

- **message** (string)

Message to show in window.

Returns

Real. Value input, (or null if cancel pressed).

Return type

Number

Example

To create an input window with title *Input* and message *Input number* and return the value input:

```
var value = Window.GetNumber("Input", "Input number");
```

GetOptions(title*[string]*, message*[string]*, options*[object]*) [static]

Description

Map a window allowing you to input various options. OK and Cancel buttons are shown.

Arguments

- **title** (string)

Title for window.

- **message** (string)

Message to show in window.

- **options** (array of objects)

Array of objects listing options that can be set. If OK is pressed the objects will be updated with the values from the widgets. If cancel is pressed they will not.

Object has the following properties:

Name	Type	Description
selected (optional)	boolean	If checkbox is selected or not
text	string	Text to show next to option
type	string	Type of option. Can be "label" (plain text), "text" (a one line text widget), "textbox" (a multi line text widget) or "checkbox" (a checkable option)
value	string	Text to show for option

Returns

false if cancel pressed, true if OK pressed.

Return type

Boolean

Example

To create a window with title *Options* , message *Please give the options* with label, text, textbox and checkbox widgets:

```
var options = [ { text:"Label example", type:"label", value:"banana" }, {
text:"Text example", type:"text", value:"single line of text" }, { text:"Textbox
example", type:"textbox",
value:"Multiple\\nlines\\nof\\ntext" }, { text:"Checkbox example",
type:"checkbox", value:"Do this?", selected:true }
                ];
ok = Window.GetOptions("Options", "Please give the options", options);
```

GetString(title*[string]*, message*[string]*) [static]

Description

Map a window allowing you to input a string. OK and Cancel buttons are shown.

Arguments

- **title** (string)

Title for window.

- **message** (string)

Message to show in window.

Returns

String. Value input, (or null if cancel pressed).

Return type

String

Example

To create an input window with title *Input* and message *Input string* and return the value input:

```
var value = Window.GetString("Input", "Input string");
```

Information(title*[string]*, info*[string]*, buttons (optional)*[constant]*) [static]

Description

Show information in a window.

Arguments

- **title** (string)

Title for window.

- **info** (string)

Information to show in window.

- **buttons (optional)** (constant)

The buttons to use. Can be bitwise OR of Window.OK, Window.CANCEL, Window.YES or Window.NO. If this is omitted an OK button will be used.

Returns

Button pressed

Return type

Number

Example

To show information *Information* in window with title *Example* with OK and Cancel buttons:

```
var answer = Window.Information("Example", "Information", Window.OK |
Window.CANCEL);
if (answer == Window.CANCEL) Message("You pressed the Cancel button");
```

Message(title*[string]*, message*[string]*, buttons (optional)*[constant]*) [static]

Description

Show a message in a window.

Arguments

- **title** (string)

Title for window.

- **message** (string)

Message to show in window.

- **buttons (optional)** (constant)

The buttons to use. Can be bitwise OR of Window.OK, Window.CANCEL, Window.YES or Window.NO. If this is omitted an OK button will be used

Returns

Button pressed

Return type

Number

Example

To show message *Press YES or NO* in window with title *Example* with YES and NO buttons:

```
var answer = Window.Message("Example", "Press YES or NO", Window.YES |
Window.NO);
if (answer == Window.NO) Message("You pressed No");
```

Question(title*[string]*, question*[string]*, buttons (optional)*[constant]*) [static]

Description

Show a question in a window.

Arguments

- **title** (string)

Title for window.

- **question** (string)

Question to show in window.

- **buttons (optional)** (constant)

The buttons to use. Can be bitwise OR of Window.OK, Window.CANCEL, Window.YES or Window.NO. If this is

omitted Yes and No button will be used.

Returns

Button pressed

Return type

Number

Example

To show question *Do you want to continue?* in window with title *Question*:

```
var answer = Window.Question("Question", "Do you want to continue?");
if (answer == Window.NO) Message("You pressed No");
```

Warning(title*[string]*, warning*[string]*, buttons (optional)*[constant]*) [static]

Description

Show a warning message in a window.

Arguments

- **title** (string)

Title for window.

- **warning** (string)

Warning message to show in window.

- **buttons (optional)** (constant)

The buttons to use. Can be bitwise OR of Window.OK, Window.CANCEL, Window.YES or Window.NO. If this is omitted an OK button will be used.

Returns

Button pressed

Return type

Number

Example

To show warning *Title is blank\nSet to ID?* in window with title *Warning* with Yes and No buttons:

```
var answer = Window.Warning("Warning", "Title is blank\nSet to ID?", Window.YES
| Window.NO);
if (answer == Window.NO) Message("You pressed No");
```