



Scripting User Guide

Legal Notices

Published by Toon Boom Animation Inc.

Corporate Headquarters

7 Laurier Avenue East
Montreal, Quebec
Canada H2T 1E4
Tel: (514) 278-8666
Fax: (514) 278-2666
toonboom.com

Disclaimer

The content of this manual is covered by a specific limited warranty and exclusions and limit of liability under the applicable License Agreement as supplemented by the special terms and conditions for Adobe® Flash® File Format (SWF). Please refer to the License Agreement and to those special terms and conditions for details.

The content of this manual is the property of Toon Boom Animation Inc. and is copyrighted. Any reproduction in whole or in part is strictly prohibited.

For additional copies of this manual, please contact Toon Boom Animation Inc. at the Corporate Headquarters address.

Copyright © 2010 by Toon Boom Animation Inc. All rights reserved.

Trademarks

Toon Boom Animate is a trademark owned by Toon Boom Animation Inc. All other trademarks are the property of their respective owners.

Credits

Documentation Development: Peter Cawthorne

Content Development: Marie-Eve Chartrand, Anouk Whissell, Shabana Ali

Art Development: Marie-Eve Chartrand, Anouk Whissell, Shabana Ali, Tania Gray, Annie Rodrigue

Product Manager: Sébastien Miglio

Publication Date

March 2010

Contents

Chapter 1: Scripting Introduction	9
Scripting Overview	9
Scripting Template	9
Using Qt Script to Automate Toon Boom Animate Functions	9
Creating a Qt Script	10
Exporting and Importing Scripts	10
Linking a Script to a Toolbar Button	11
Animate Server Network Environment	12
Using Qt Script to Automate Toon Boom Animate Functions - server functions	12
Exporting and Importing Scripts	14
Linking a Script to a Toolbar Button	15
Scripting Reference	15
Chapter 2: Scripting Reference	17
Function Summary	18
About	35
animate	37
animatePro	37
applicationPath	37
controlCenterApp	37
demoVersion	37
educVersion	38
fullVersion	38
getApplicationPath	38
getFlavorString	38
getVersionInfoStr	38
harmony	39
interactiveApp	39
isAnimate	39
isAnimatePro	39
isControlCenterApp	39
isDemoVersion	40
isEducVersion	40
isFullVersion	40
isHarmony	40
isInteractiveApp	40
isLinuxArch	41
isMacArch	41
isMacIntelArch	41
isMacPpcArch	41
isMainApp	41
isPaintMode	42
isScanApp	42
isStage()	42
isWindowsArch	42
isXsheetMode	42
linuxArch	43
macArch	43
macIntelArch	43
macPpcArch	43
mainApp	43
paintMode	44
productName	44
scanApp	44
stage	44
windowsArch	44

xsheetMode	45
Action	46
perform	46
Column	47
add	48
clearKeyFrame	48
getColorForXSheet	48
getColumnListOfType	48
getCurrentVersionFor Drawing	49
getDisplayname	49
getDrawingColumnList	49
getDrawingName	49
getDrawingTimings	49
getElementIdOfDrawing	50
getEntry	50
getName	50
getNextKeyDrawing	50
getTextOfExpr	51
importSound	51
isKeyFrame	51
numberOf	51
rename	52
setColorForXSheet	52
setElementIdOfDrawing	52
setEntry	53
setKeyFrame	53
setTextOfExpr	53
type	54
CopyPaste	55
createTemplateFromSelection	55
pasteTemplateIntoScene	55
usePasteSpecial	56
setPasteSpecialCreateNewColumn	56
setPasteSpecialElementTimingColumnMode	57
setPasteSpecialAddRemoveMotionKeyFrame	57
setPasteSpecialAddRemoveVelocityKeyFrame	57
setPasteSpecialAddRemoveAngleKeyFrame	57
setPasteSpecialAddRemoveSkewKeyFrame	57
setPasteSpecialAddRemoveScalingKeyFrame	58
setPasteSpecialForcesKeyFrameAtBegAndEnd	58
setPasteSpecialOffsetKeyFrames	58
setPasteSpecialReplaceExpressionColumns	58
setPasteSpecialDrawingAction	58
setPasteSpecialDrawingFileMode	59
setPasteSpecialDrawingAutomaticExtendExposure	59
setPasteSpecialColorPaletteOption	59
Element	60
add	60
fieldChart	60
Folder	61
getNameById	61
id	61
numberOf	61
pixmapFormat	61
remove	62
renameById	62
scanType	62
vectorType	62

Exporter	63
cleanExportDir	63
getExportDir	63
Frame	64
current	64
insert	65
numberOf	65
remove	65
setCurrent	65
Function Curve	66
addCtrlPointAfter3DPath	68
addKeyFrame3DPath	68
angleEaseIn	68
angleEaseOut	69
holdStartFrame	69
holdStep	69
holdStopFrame	69
numberOfPoints	70
numberOfPoints3DPath	70
pointBias3DPath	70
pointConstSeg	70
pointContinuity	71
pointContinuity3DPath	71
pointEaseIn	71
pointEaseOut	71
pointHandleLeftX	72
pointHandleLeftY	72
pointHandleRightX	72
pointHandleRightY	72
pointLockedAtFrame	73
pointTension3DPath	73
pointX	73
pointX3DPath	73
pointY	74
pointY3DPath	74
pointZ3DPath	74
removePoint3DPath	74
setBezierPoint	75
setEasePoint	75
setHoldStartFrame	76
setHoldStep	76
setHoldStopFrame	76
setPoint3DPath	76
setVeloBasedPoint	77
MessageLog	78
debug	78
isDebug	78
setDebug	78
trace	78
Node	79
add	81
addCompositeToGroup	81
coordX	81
coordY	81
createGroup	82
deleteNode	82
dstNode	82
equals	82

explodeGroup	83
flatDstNode	83
flatSrcNode	83
getCameras	83
getDefaultCamera	84
getEnable	84
getMatrix	84
getName	84
getTextAttr	84
isGroup	85
isLinked	85
link	85
linkAttr	85
linkedColumn	86
noNode	86
numberOfInputPorts	86
numberOfOutputLinks	86
numberOfOutputPorts	87
numberOfSubNodes	87
parentNode	87
rename	87
root	88
setAsDefaultCamera	88
setAsGlobalDisplay	88
setCoord	88
setEnable	88
setGlobalToDisplayAll	89
setTextAttr	89
srcNode	89
subNode	89
subNodeByName	90
type	90
unlink	90
unlinkAttr	90
width/height	91
PaletteManager	92
getCurrentColorId	92
getCurrentColorName	92
getCurrentPaletteld	92
getCurrentPaletteName	93
setCurrentPaletteByld	93
setCurrentColorByld	93
setCurrentPaletteAndColorByld	93
getCurrentPaletteSize	93
getColorName	94
getColorId	94
getNumPalettes	94
getNumPalettes	94
getPaletteName	94
getPaletteName	95
getPaletteld	95
getPaletteld	95
PenstyleManager	96
getNumberOfPenstyles	97
getPenstyleName	97
getCurrentPenstyleName	97
setCurrentPenstyleByName	97
setCurrentPenstyleByIndex	97

changeCurrentPenstyleMinimumSize	98
changeCurrentPenstyleMaximumSize	98
changeCurrentPenstyleOutlineSmoothness	98
changeCurrentPenstyleCenterlineSmoothness	98
changeCurrentPenstyleEraserFlag	98
getCurrentPenstyleIndex	99
getCurrentPenstyleMinimumSize	99
getCurrentPenstyleMaximumSize	99
getCurrentPenstyleOutlineSmoothness	99
getCurrentPenstyleCenterlineSmoothness	99
getCurrentPenstyleEraserFlag	100
exportPenstyleToString	100
exportPenstyleListToString	100
importPenstyleListFromString	100
savePenstyles	100
Preferences	101
getBool	101
getColor	102
getDouble	102
getInt	102
getString	102
setBool	102
setColor	103
setDouble	103
setInt	103
setString	103
Render	104
frameReady	105
renderFinished	105
setCombine	105
setFieldType	105
setBgColor	106
setResolution	106
setRenderDisplay	106
setWriteEnabled	106
renderScene	106
renderSceneAll	107
cancelRender	107
Scene	108
beginUndoRedoAccum	109
cancelUndoRedoAccum	109
clearHistory	109
coordAtCenterX	110
coordAtCenterY	110
currentEnvironment	110
currentJob	110
currentProjectPath	110
currentProjectPathRemapped	111
currentResolutionX	111
currentResolutionY	111
currentScene	111
currentVersion	111
defaultResolutionFOV	112
defaultResolutionName	112
defaultResolutionX	112
defaultResolutionY	112
endUndoRedoAccum	112
getFrameRate	113

fromOGL	113
getCameraMatrix	113
numberOfUnitsX	113
numberOfUnitsY	113
numberOfUnitsZ	114
saveAll	114
saveAsNewVersion	114
setCoordAtCenter	114
setNumberOfUnits	114
setUnitsAspectRatio	115
toOGL	115
unitsAspectRatioX	115
unitsAspectRatioY	115
setDefaultResolution	115
setFrameRate	116
Selection	117
clearSelection	118
addDrawingColumnToSelection	118
addColumnToSelection	118
addNodeToSelection	118
extendSelectionWithColumn	118
numberOfCellColumnsSelected	119
numberOfFramesSelected	119
numberOfNodesSelected	119
selectAll	119
selectedCellColumn	119
selectedNode	120
setSelectionFrameRange	120
Sound	121
setSampleRate	121
setChannelSize	121
setChannelCount	121
getSoundtrack	122
getSoundtrackAll	122
SpecialFolders	123
app	123
bin	124
config	124
etc	124
htmlHelp	124
lang	124
library	125
pdf	125
platform	125
plugins	125
resource	125
root	126
temp	126
userConfig	126
Timeline	127
firstFrameSel	129
isAncestorOf	129
layerIsColumn	129
layerIsNode	129
layerToColumn	130
layerToNode	130
numFrameSel	130
numLayers	130

numLayerSel	130
parentNodeIndex	131
sellsColumn	131
sellsNode	131
selToColumn	131
selToLayer	132
selToNode	132
setDisplayToUnconnected	132
View	133
column	133
currentView	133
group	133
refreshViews	134
type	134

Chapter 1

Scripting Introduction

In the scripting section you will find information which helps you to understand how to use Qt®, the scripting language supported by Toon Boom Animate.

This section of the Animate 2 Scripting User Guide is divided into two parts:

- Scripting Overview, on page 9
- Scripting Reference, on page 15

Scripting Overview

The following topics are addressed:

- Scripting Template, on page 9
- Using Qt Script to Automate Toon Boom Animate Functions, on page 9
- Creating a Qt Script, on page 10
- Exporting and Importing Scripts, on page 10
- Linking a Script to a Toolbar Button, on page 11
- Animate Server Network Environment, on page 12

Qt Script provides access to many of the functions supported in the Toon Boom Animate interface. With Qt Script, you can automate a number of Toon Boom Animate functions to speed the completion of various repetitive tasks.

You can use QSA Workbench to create Qt scripts for Toon Boom Animate.

To use scripts prepared by other users or for other scenes, you must first export the scripts from the originating scene and then import the scripts into a target scene.

You can add buttons to the Scripting Tools toolbar so that you can access them easily. They will appear to the right of the default Scripting Toolbar buttons.

Scripting Template

To find out the names for the different layers and parameters available in Animate, you should do the following:

1. Download the sample Scripting template available in the Documentation section of the Toon Boom website at: <http://www.toonboom.com/products/animate/eLearning/>
2. Once you have downloaded the package, uncompress it and read the instructions contained in the package.

Using Qt Script to Automate Toon Boom Animate Functions

Qt Script provides access to many of the functionalities supported in the Toon Boom Animate interface.

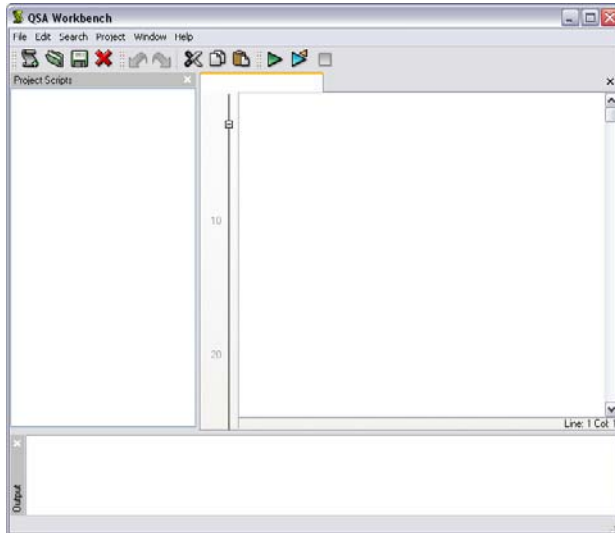
Qt Script is an object-oriented scripting language based on the ECMAScript standard, like JavaScript and JScript. However, there are some differences that distinguish it from these scripting languages, which are familiar to web programmers.


Creating a Qt Script

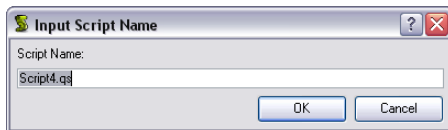
Use QSA Workbench to create Qt scripts for Toon Boom Animate.

To create a script:

1. Click on the **Edit Script**  button in the Scripting Tools toolbar. The QSA Workbench dialog box opens.



2. Select **File > New Script** . The Input Script Name dialog box appears.




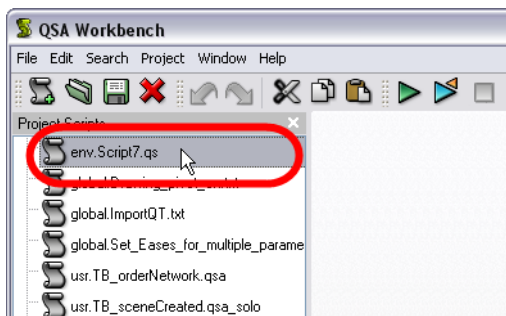
3. Enter a name in the **Script Name** field and click on **OK**.


Exporting and Importing Scripts

To use scripts prepared by other users or for other scenes, you must first export the scripts from the originating scene and then import the scripts into a target scene.



To export a script:

1. In Toon Boom Animate, open the scene that you want to export the scripts from.
2. Click on the **Edit Script**  button to open QSA Workbench.
3. Double-click the script that you want to export from the Project Scripts list, this will load it into the right hand side of the interface.



4. Select **File > Export Script** . The Export Script dialog box opens.
The **Export Script** window opens.
5. Use the Export Script dialog box to select a name and location for the exported script and then click **OK**.
The script is saved. It is now ready to be imported into other scenes.

To import a script:

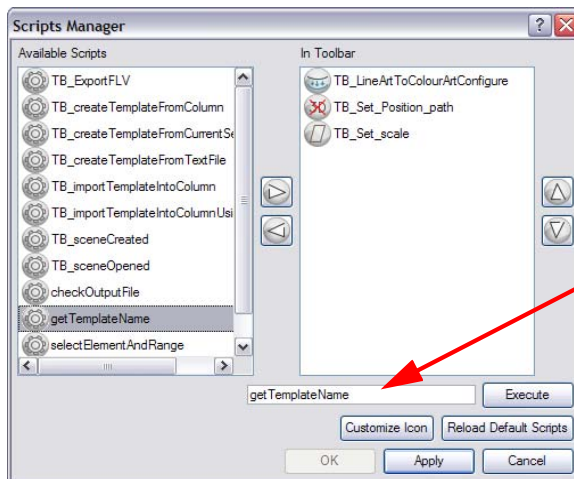
1. In Toon Boom Animate, open the scene that you want to import the scripts into.
2. Click on the **Edit Script**  button to open QSA Workbench.
3. Select **File > Import Script** .
4. Use the Import Script dialog box to locate the script file that you want to import. Select a script and click on **Open**.
The script is imported and appears in the Project Scripts list. It is automatically saved at the user level (all scripts are `usr.Scriptname.qs`).

Linking a Script to a Toolbar Button


You can add buttons to the Scripting Tools toolbar so that you can access them easily. They will appear to the right of the default Scripting Toolbar buttons.

To link a script to a toolbar button:

1. Click on the **Manage Scripts**  button on the Scripting Tools toolbar. A dialog box containing all of the available scripts opens.



You can select a script from this dialog box and run it immediately by clicking on the Execute button. The field displays the name of the currently selected script.

2. In the Available Scripts list, select the script you want to link to a toolbar button.
3. Click on the **Right Arrow**  button to add the script to the In Toolbar list.
4. Click on **OK** to add the button and close the dialog box.

Animate Server Network Environment

When you are working in a Network environment, use these procedures.

- Using Qt Script to Automate Toon Boom Animate Functions - server functions, on page 12
- Exporting and Importing Scripts, on page 14
- Linking a Script to a Toolbar Button, on page 15

Using Qt Script to Automate Toon Boom Animate Functions - server functions

Qt Script provides access to many of the functionalities supported in the Toon Boom Animate interface.

Qt Script is an object-oriented scripting language based on the ECMAScript standard, like JavaScript and JScript. However, there are some differences that distinguish it from these scripting languages, which are familiar to web programmers.

Creating a Qt Script

Use QSA Workbench to create Qt scripts for Toon Boom Animate.

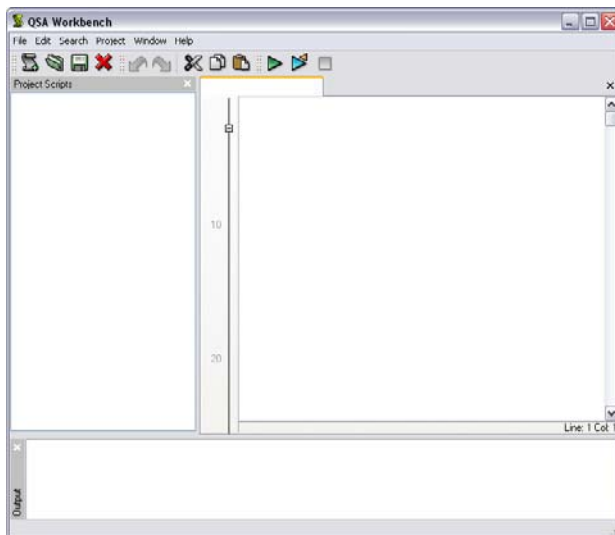
By default, all scripts are saved in:


- **On Mac OS X**
`/Users/Current_User_Name/Library/Preferences/Toon Boom Animation/Toon Boom Animate/780-scripts/Script.qsa`
- **On Windows 7/ Vista**
`C:\Users\Current_User_Name\AppData\Roaming\Toon Boom Animation\Toon Boom Animate\780-scripts`
- **On Windows XP**
`C:\Documents and Settings\Current_User_Name\Application Data\Toon Boom Animation\Toon Boom Animate\780-scripts`

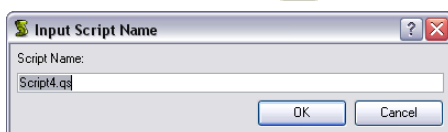
To create a script:

1. Click on the **Edit Script**  button in the Scripting Tools toolbar.

The QSA Workbench dialog box opens.

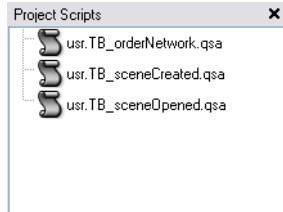


2. Select **File > New Script** . The Input Script Name dialog box appears.



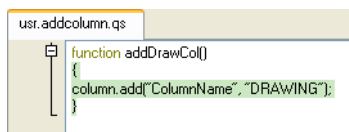
3. Enter a name in the **Script Name** field and click on **OK**.

The name of your script appears in the Project Scripts panel.

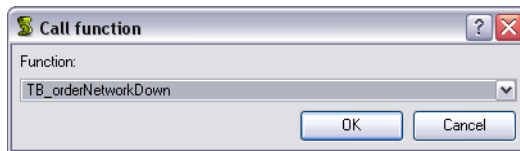


4. Double-click on the script name in the Project Scripts list panel. A tab with a text box appears to the right of the Projects Scripts panel.
5. Write your script in the tabbed text box. Try the following script:

```
function add3dPathCol ()
{
column.add ("ColumnName", "3DPATH");
}
```

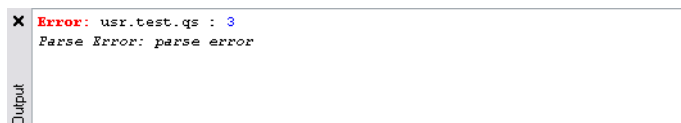


6. To test your script, click on the green **Play** button.
 - The Call Function dialog box appears.



If you select the script and click on **OK**, a new 3Dpath column will appear in your Xsheet View with the name ColumnName.


- If a syntax error occurs, it will be displayed in the Output text box.

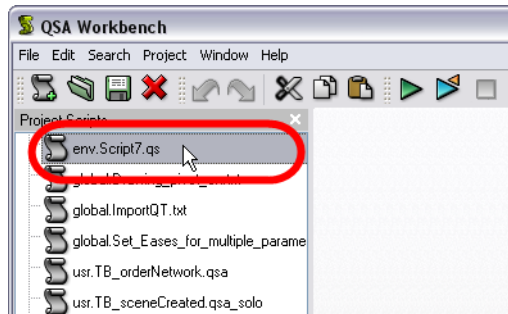



Exporting and Importing Scripts

To use scripts prepared by other users or for other scenes, you must first export the scripts from the originating scene and then import the scripts into a target scene.



To export a script:

1. In Toon Boom Animate, open the scene that you want to export the scripts from.
2. Click on the **Edit Script**  button to open QSA Workbench.
3. Select the script that you want to export from the Project Scripts list.



4. Select **File > Export Script** . The Export Script dialog box opens.
The **Export Script** window opens.
5. Use the Export Script dialog box to select a name and location for the exported script and then click **OK**.
The script is saved. It is now ready to be imported into other scenes.

To import a script:

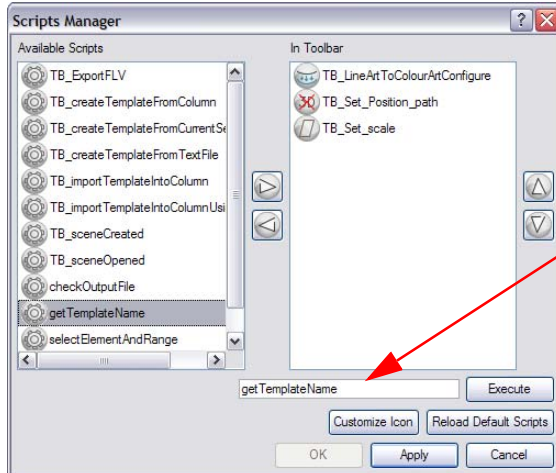
1. In Toon Boom Animate, open the scene that you want to import the scripts into.
2. Click on the **Edit Script**  button to open QSA Workbench.
3. Select **File > Import Script** .
4. To save the script:
 - ▶ Click on the **User** button to store the script into the user profile so the user can always access the script.
5. Use the Import Script dialog box to locate the script file that you want to import. Select a script and click on **Open**.
The script is imported and appears in the Project Scripts list.

Linking a Script to a Toolbar Button


You can add buttons to the Scripting Tools toolbar so that you can access them easily. They will appear to the right of the default Scripting Toolbar buttons.

To link a script to a toolbar button:

1. Click on the **Manage Scripts**  button on the Scripting Tools toolbar. A dialog box containing all of the available scripts opens.



You can select a script from this dialog box and run it immediately by clicking on the Execute button. The field displays the name of the currently selected script.

2. In the Available Scripts list, select the script you want to link to a toolbar button.
3. Click on the **Right Arrow**  button to add the script to the In Toolbar list.
4. Click on **OK** to add the button and close the dialog box.

Scripting Reference

In the Scripting Reference chapter you will find a detailed alphabetical list of all the built-in Toon Boom Animate objects and the functions which you can access with Qt Script. Refer to Scripting Reference, on page 17.

Chapter 2

Scripting Reference

This chapter describes the built-in objects which can be accessed with Qt® Script.

With Qt® Script, you can access the following built-in Toon Boom Animate objects.

- About, on page 35
- Action, on page 46
- Column, on page 47
- CopyPaste, on page 55
- Element, on page 60
- Exporter, on page 63
- Frame, on page 64
- Function Curve, on page 66
- MessageLog, on page 78
- Node, on page 79
- PaletteManager, on page 92
- PenstyleManager, on page 96
- Preferences, on page 101\
- Render, on page 104
- Scene, on page 108
- Selection, on page 117
- Sound, on page 121
- SpecialFolders, on page 123
- Timeline, on page 127
- View, on page 133

Each object has its own set of functions that can manipulate the attributes of the object.

All functions can receive the following data types as arguments. All functions return the same data types.

- An integer (numerical value)
On failure, the function returns -1.
- A string
When used as arguments, strings must be enclosed in quotes. On failure the function returns the null string "".
- A boolean value (true or false).
The function returns false on failure.

As arguments, these data types can be supplied in the function or can be generated by other functions.

For more information on the Qt scripting language, see the following:

- **Language:** <http://doc.trolltech.com/qs-a-1.1.5/language.html>
- **Creating Qt Scripts, including building dialog boxes:**
<http://doc.trolltech.com/qs-a-1.1.5/qs-a-5.html>

Function Summary

These tables show all of the supported functions, their arguments and descriptions, each description provided in the function tables includes a link to the page in this document where you can find more detail.

- [About Function](#), on page 18
- [Action Function](#), on page 20
- [Column Function](#), on page 20
- [CopyPaste Function](#), on page 21
- [Element Function](#), on page 22
- [Exporter Function](#), on page 22
- [Frame Function](#), on page 23
- [Function Curve Function](#), on page 23
- [MessageLog Function](#), on page 25
- [Node Function](#), on page 25
- [PaletteManager Function](#), on page 27
- [PenStyleManager Function](#), on page 28
- [Preferences Function](#), on page 29
- [Render Function](#), on page 29
- [Scene Function](#), on page 30
- [Selection Function](#), on page 31
- [Sound Function](#), on page 32
- [SpecialFolders Function](#), on page 32
- [Timeline Function](#), on page 33
- [View Function](#), on page 34

About Function	Description
<code>about.animate</code>	This is a read-only property that is true whenever this script is being executed by Animate For details, see animate , on page 37.
<code>about.animatePro</code>	This is a read-only property that is true whenever this script is being executed by Animate Pro. For details, see animatePro , on page 37.
<code>about.applicationPath;</code>	Contains application path. For details, see applicationPath , on page 37.
<code>about.controlCenterApp;</code>	Contains true when ControlCenter. For details, see controlCenterApp , on page 37.
<code>about.demoVersion;</code>	Contains true when a Demo Version. For details, see demoVersion , on page 37.
<code>about.educVersion;</code>	Contains true when in Educational Version. For details, see educVersion , on page 38.
<code>about.fullVersion;</code>	Contains true when in Commercial Version. For details, see fullVersion , on page 38.
<code>about.getApplicationPath();</code>	Returns the application path. For details, see getApplicationPath , on page 38.
<code>about.getFlavorString();</code>	Returns the flavour string. For details, see getFlavorString , on page 38.
<code>about.getVersionInfoStr();</code>	Returns the version string. For details, see getVersionInfoStr , on page 38.
<code>about.harmony;</code>	Returns true when in Harmony. For details, see harmony , on page 39.
<code>about.interactiveApp</code>	Contains true when interactive application. For details, see interactiveApp , on page 39.
<code>about.isAnimate()</code>	Returns true when in Animate. For details, see isAnimate , on page 39.

About Function	Description
<code>about.isAnimatePro()</code>	Returns true when in Animate Pro. For details, see <code>isAnimatePro</code> , on page 39.
<code>about.isControlCenterApp()</code> ;	Returns true when in ControlCenter. For details, see <code>isControlCenterApp</code> , on page 39.
<code>about.isDemoVersion()</code> ;	Returns true when in a demo version. For details, see <code>isDemoVersion</code> , on page 40.
<code>about.isEducVersion()</code> ;	Returns true when in Educational Version. For details, see <code>isEducVersion</code> , on page 40.
<code>about.isFullVersion()</code> ;	Returns true when in Commercial Version. For details, see <code>isFullVersion</code> , on page 40.
<code>about.isHarmony()</code> ;	This is a read-only property that is true whenever this script is being executed by Harmony. For details, see <code>isHarmony</code> , on page 40.
<code>about.isInteractiveApp()</code> ;	Returns true when in Interactive. For details, see <code>isInteractiveApp</code> , on page 40.
<code>about.isLinuxArch()</code> ;	Returns true when on Linux OS. For details, see <code>isLinuxArch</code> , on page 41.
<code>about.isMacArch()</code> ;	Returns true when on OS X. For details, see <code>isMacArch</code> , on page 41.
<code>about.isMacIntelArch()</code> ;	Returns true when on OS X, Intel. For details, see <code>isMacIntelArch</code> , on page 41.
<code>about.isMacPpcArch()</code> ;	Returns true when on OS X, PowerPC. For details, see <code>isMacPpcArch</code> , on page 41.
<code>about.isMainApp()</code> ;	Returns true when in Stage/DigitalPro. For details, see <code>isMainApp</code> , on page 41.
<code>about.isPaintMode()</code> ;	Returns true when in Paint Mode. For details, see <code>isPaintMode</code> , on page 42.
<code>about.isScanApp()</code> ;	Returns true when in TbScan. For details, see <code>isScanApp</code> , on page 42.
<code>about.isStage()</code> ;	Returns true when in Stage. For details, see <code>isStage()</code> , on page 42.
<code>about.isWindowsArch()</code> ;	Returns true when on Windows. For details, see <code>isWindowsArch</code> , on page 42.
<code>about.isXsheetMode()</code> ;	Returns true when in XSheet Mode. For details, see <code>isXsheetMode</code> , on page 42.
<code>about.linuxArch;</code>	Contains true when on Linux. For details, see <code>linuxArch</code> , on page 43.
<code>about.macArch;</code>	Contains true when on OS X. For details, see <code>macArch</code> , on page 43.
<code>about.macIntelArch;</code>	Contains true when on OS X, Intel. For details, see <code>macIntelArch</code> , on page 43.
<code>about.macPpcArch;</code>	Contains true when on OS X, PowerPC. For details, see <code>macPpcArch</code> , on page 43.
<code>about.mainApp;</code>	Contains true when in Stage, DigitalPro. For details, see <code>mainApp</code> , on page 43.
<code>about.paintMode;</code>	Contains true when in PaintMode. For details, see <code>paintMode</code> , on page 44.
<code>about.productName;</code>	Returns the product name. For details, see <code>productName</code> , on page 44.
<code>about.scanApp;</code>	Contains true when in TbScan. For details, see <code>scanApp</code> , on page 44.
<code>about.stage;</code>	Returns true when in Stage. For details, see <code>stage</code> , on page 44.

About Function	Description
<code>about.windowsArch;</code>	Contains true when on Windows. For details, see <code>windowsArch</code> , on page 44.
<code>about.xsheetMode;</code>	Contains true when in XSheet Mode. For details, see <code>xsheetMode</code> , on page 45.
Action Function	Description
<code>action.perform(name);</code>	This function calls the menu function directly. For details, see <code>perform</code> , on page 46.
Column Function	Description
<code>column.add("columnName", "columnType");</code>	Adds a column with the specified name and type. For details, see <code>add</code> , on page 48.
<code>column.clearKeyFrame("columnName", atFrame);</code>	Removes a keyframe from a cell in a column. For details, see <code>clearKeyFrame</code> , on page 48.
<code>column.getColorForXSheet();</code>	Returns the colour of a specified XSheet column. For details, see <code>getColorForXSheet</code> , on page 48.
<code>column.getColumnListOfType("DRAWING");</code>	Returns the name of all drawing columns For details, see <code>getColumnListOfType</code> , on page 48.
<code>column.getCurrentVersionForDrawing();</code>	Returns the current drawing version for the given column and drawing name. For details, see <code>getCurrentVersionForDrawing</code> , on page 49.
<code>column.getDisplayName();</code>	Returns the display name of the column. Note that the column functions now return a unique identifier for the columns. In order to translate this unique identifier into the same string that is viewable in the xsheet, this function must be used. For details, see <code>getDisplayName</code> , on page 49.
<code>column.getDrawingName();</code>	Returns the name at given frame of a drawing column. For details, see <code>getDrawingName</code> , on page 49.
<code>column.getDrawingTimings("columnName", true);</code>	Lists list of all drawing names used in the drawing column For details, see <code>getDrawingTimings</code> , on page 49.
<code>column.getElementIdOfDrawing("columnName");</code>	Returns the id of the element linked to the specified drawing column. For details, see <code>getElementIdOfDrawing</code> , on page 50.
<code>column.getEntry("columnName", subColumn, atFrame);</code>	Returns the value of a cell in a column. For details, see <code>getEntry</code> , on page 50.
<code>column.getName(columnNumber);</code>	Returns the unique identifier that names the column. This is not the display name shown in the xsheet view. For details, see <code>getName</code> , on page 50.
<code>column.getTextOfExpr("columnName");</code>	Returns the expression text in the identified column. For details, see <code>getTextOfExpr</code> , on page 51.
<code>column.importSound();</code>	Import a external sound file into a sound column. For details, see <code>importSound</code> , on page 51.
<code>column.isKeyFrame("columnName", subColumn, atFrame);</code>	Returns true or false indicating if a cell in a column is a keyframe. For details, see <code>isKeyFrame</code> , on page 51.
<code>column.getNextKeyDrawing();</code>	Returns the frame number of the next key drawing in the given drawing column from the specified start frame. For details, see <code>getNextKeyDrawing</code> , on page 50.
<code>column.numberOf();</code>	Returns the number of columns in the scene. For details, see <code>numberOf</code> , on page 51.

Column Function	Description
<code>column.rename("oldName", "newName");</code>	Renames the specified column. For details, see rename , on page 52.
<code>column.setColorForXSheet();</code>	Change the colour of an XSheet column. For details, see setColorForXSheet , on page 52.
<code>column.setElementIdOfDrawing("columnName", "elementId");</code>	Links an empty Drawing column to an element. For details, see setElementIdOfDrawing , on page 52.
<code>column.setEntry("columnName", subColumn, atFrame, "value");</code>	Set the value of a cell in a column. For details, see setEntry , on page 53.
<code>column.setKeyFrame("columnName", atFrame);</code>	Makes a cell in a column a keyframe. For details, see setKeyFrame , on page 53.
<code>column.setTextOfExpr("columnName", "text");</code>	Sets the value in the Expression column to the specified text. For details, see setTextOfExpr , on page 53.
<code>column.type("columnName");</code>	This function returns the column type. There are nine column types: Drawing (DRAWING), Sound (SOUND), 3D Path (3DPATH), Bezier Curve (BEZIER), Ease Curve (EASE), Expression (EXPR), Timing (TIMING) for timing columns, Quaternion path (QUATERNIONPATH) for 3D rotation and Annotation (ANNOTATION) for annotation columns. For details, see type , on page 54.

CopyPaste Function	Description
<code>copyPaste.createTemplateFromSelection("name", const "path");</code>	This function creates template from the current selection in the scene, using only the current drawing versions. For details, see createTemplateFromSelection , on page 55.
<code>copyPaste.pasteTemplateIntoScene(const "templateSrcPath", "insertColumnName", int insertFrame);</code>	This function pastes the template into the scene. For details, see pasteTemplateIntoScene , on page 55.
<code>copyPaste.usePasteSpecial(flag);</code>	This function enables the paste special options. For details, see pasteTemplateIntoScene , on page 55.
<code>copyPaste.setExtendScene(flag);</code>	This function sets the option to extend the scene to accommodate the incoming template. For details, see pasteTemplateIntoScene , on page 55.
<code>copyPaste.setPasteSpecialCreateNewColumn(flag);</code>	This function sets the option to create new columns to paste. For details, see setPasteSpecialCreateNewColumn , on page 56.
<code>copyPaste.setPasteSpecialElementTimingColumnMode("mode");</code>	This function sets the paste special element timing mode for calls to pasteTemplateIntoScene . For details, see setPasteSpecialElementTimingColumnMode , on page 57.
<code>copyPaste.setPasteSpecialAddRemoveMotionKeyFrame(flag);</code>	PEGS: This function is a paste special option for pegs and functions. For details, see setPasteSpecialAddRemoveMotionKeyFrame , on page 57.
<code>copyPaste.setPasteSpecialAddRemoveVelocityKeyFrame(flag);</code>	PEGS: This function is a paste special option for pegs and functions. For details, see setPasteSpecialAddRemoveVelocityKeyFrame , on page 57.
<code>copyPaste.setPasteSpecialAddRemoveAngleKeyFrame(flag);</code>	PEGS: This function is a paste special option for pegs and functions. For details, see setPasteSpecialAddRemoveAngleKeyFrame , on page 57.
<code>copyPaste.setPasteSpecialAddRemoveSkewKeyFrame(flag);</code>	PEGS: This function is a paste special option for pegs and functions. For details, see setPasteSpecialAddRemoveScalingKeyFrame , on page 58.
<code>copyPaste.setPasteSpecialAddRemoveScalingKeyFrame(flag);</code>	PEGS: This function is a paste special option for pegs and functions. For details, see setPasteSpecialAddRemoveScalingKeyFrame , on page 58.

CopyPaste Function	Description
<code>copyPaste.setPasteSpecialForcesKeyFrameAtBegAndEnd(flag);</code>	PEGS: This function is a paste special option for pegs and functions. For details, see <code>setPasteSpecialForcesKeyFrameAtBegAndEnd</code> , on page 58.
<code>copyPaste.setPasteSpecialOffsetKeyFrames(flag);</code>	PEGS: This function is a paste special option for pegs and functions. For details, see <code>setPasteSpecialOffsetKeyFrames</code> , on page 58.
<code>copyPaste.setPasteSpecialReplaceExpressionColumns(flag);</code>	PEGS: This function is a paste special option for pegs and functions. For details, see <code>setPasteSpecialReplaceExpressionColumns</code> , on page 58.
<code>copyPaste.setPasteSpecialDrawingAction("mode");</code>	DRAWING: This function sets the drawing file mode - only used if the <code>DrawingAction</code> is set to <code>ADD_OR_REMOVE_EXPOSURE</code> For details, see <code>setPasteSpecialDrawingAction</code> , on page 58.
<code>copyPaste.setPasteSpecialDrawingAutomaticExtendExposure(extendExposure, keyFrameMode);</code>	DRAWING: This function sets the drawing file mode For details, see <code>setPasteSpecialDrawingFileMode</code> , on page 59.
<code>copyPaste.setPasteSpecialColorPaletteOption("mode");</code>	This function sets the colour palette option. For details, see <code>setPasteSpecialColorPaletteOption</code> , on page 59.

Element Function	Description
<code>element.add("name", "scanType", "fieldChart", "fileFormat", "Vectorize");</code>	Adds an element to the scene and returns the element id of the newly added element if successful, otherwise it returns -1. For details, see <code>add</code> , on page 60.
<code>element.fieldChart();</code>	Returns the field chart size of the provided element. For details, see <code>fieldChart</code> , on page 60.
<code>element.folder(id);</code>	Returns the name of the folder on disk of the element. For details, see <code>Folder</code> , on page 61.
<code>element.getNameById(id);</code>	Returns the name of the element. For details, see <code>getNameById</code> , on page 61.
<code>element.id(index);</code>	Returns the id (key) of the element. For details, see <code>id</code> , on page 61.
<code>element.numberOf();</code>	Returns the number of elements in the scene. For details, see <code>numberOf</code> , on page 61.
<code>element.pixmapFormat();</code>	Returns the pixmap format for the provided element. For details, see <code>pixmapFormat</code> , on page 61.
<code>element.remove();</code>	Remove an element. Optionally delete all files associated with that element. For details, see <code>remove</code> , on page 62.
<code>element.renameById();</code>	Rename an element. For details, see <code>renameById</code> , on page 62.
<code>element.scanType();</code>	Returns the scan type of an element. For details, see <code>scanType</code> , on page 62.
<code>element.vectorType();</code>	Returns the vector type of an element. Will return 0 if image, or 2 if TVG. For details, see <code>vectorType</code> , on page 62.

Exporter Function	Description
<code>exporter.cleanExportDir();</code>	This function removes all files from the export directory. For details, see <code>cleanExportDir</code> , on page 63.
<code>exporter.getExportDir();</code>	This function returns the path of the projects default export directory. For details, see <code>getExportDir</code> , on page 63.

Frame Function	Description
<code>frame.current()</code> ;	Returns the number of the current frame. For details, see current , on page 64.
<code>frame.insert(atFrame, nbFrame)</code> ;	Inserts frames at the specified frame number. For details, see insert , on page 65.
<code>frame.numberOf()</code> ;	Returns the number of frames in the scene. For details, see numberOf , on page 65.
<code>frame.remove(startFrame, nbFrame)</code> ;	Removes frames from the specified frame. For details, see remove , on page 65.
<code>frame.setCurrent(frame)</code> ;	Set the current frame in a scene. For details, see setCurrent , on page 65

Function Curve Function	Description
<code>func.holdStartFrame("columnName")</code> ;	Returns the Start value from the Hold Value Editor dialog box, for Bezier, Ease and Velo-based Function Editors. For details, see holdStartFrame , on page 69.
<code>func.holdStep("columnName")</code> ;	Returns the Step value from the Hold Value Editor dialog box, for Bezier, Ease and Velo-based Function Editors. For details, see holdStep , on page 69.
<code>func.holdStopFrame("columnName")</code> ;	Returns the Stop value from the Hold Value Editor dialog box, for Bezier, Ease and Velo-based Function Editors. For details, see holdStopFrame , on page 69.

Functions common to Bezier, Ease and Velocity-Based Function Curves

<code>func.numberOfPoints("columnName")</code> ;	Returns the number of keyframes and control points on a curve. For details, see numberOfPoints , on page 70.
<code>func.pointConstSeg("columnName", point)</code> ;	Returns a 1 to indicate that the point is on a constant segment, or a 0 to indicate that the point is not on a constant segment. For details, see pointConstSeg , on page 70.
<code>func.pointContinuity("columnName", point)</code> ;	Returns the continuity of the curve that follows the point. One of the following values will be returned, in upper-case: SMOOTH, CORNER or STRAIGHT. For details, see pointContinuity , on page 71.
<code>func.pointX("columnName", point)</code> ;	Returns the X value (frame number) of a point on a function curve. For details, see pointX , on page 73.
<code>func.pointY("columnName", point)</code> ;	Returns the Y value of a point on a function curve. For details, see pointY , on page 74.

Bezier Functions

<code>func.pointHandleLeftX("columnName", point)</code> ;	Returns the X value of the left handle of a point on a curve. For details, see pointHandleLeftX , on page 72.
<code>func.pointHandleLeftY("columnName", point)</code> ;	Returns the Y value of the left handle of a point on a curve. For details, see pointHandleLeftY , on page 72.
<code>func.pointHandleRightX("columnName", point)</code> ;	Returns the X value of the right handle of a point on a curve. For details, see pointHandleRightX , on page 72.
<code>func.pointHandleRightY("columnName", point)</code> ;	Returns the Y value of the right handle of a point on a curve. For details, see pointHandleRightY , on page 72.

Ease Functions

<code>func.angleEaseIn("columnName", point)</code> ;	Returns the angle of the ease-in handle. For details, see angleEaseIn , on page 68.
--	---

Function Curve Function	Description
<code>func.angleEaseOut("columnName", point);</code>	Returns the angle of the ease-out handle. For details, see angleEaseOut , on page 69.
<code>func.pointEaseIn("columnName", point);</code>	Returns the number of frames in the ease-in. For details, see pointEaseIn , on page 71.
<code>func.pointEaseOut("columnName", point);</code>	Returns the number of frames in the ease-out. For details, see pointEaseOut , on page 71.
3D Path Functions	
<code>func.numberOfPoints3DPath("columnName");</code>	Returns the number of keyframes and control points on the 3D Path. For details, see numberOfPoints3DPath , on page 70.
<code>func.pointBias3DPath("columnName", point);</code>	Returns the bias value for the specified point on the 3D Path. For details, see pointBias3DPath , on page 70.
<code>func.pointContinuity3DPath("columnName", point);</code>	Returns the continuity value (STRAIGHT, SMOOTH or CORNER) for the specified point on the 3D Path. For details, see pointContinuity3DPath , on page 71.
<code>func.pointLockedAtFrame("columnName", point);</code>	Returns the frame the point is locked at, or 0 if the point is not locked. For details, see pointLockedAtFrame , on page 73.
<code>func.pointTension3DPath("columnName", point);</code>	Returns the tension value for the specified point on the 3D Path. For details, see pointTension3DPath , on page 73.
<code>func.pointX3DPath("columnName", point);</code>	Returns the value of the specified point on the X path. For details, see pointX3DPath , on page 73.
<code>func.pointY3DPath("columnName", point);</code>	Returns the value of the specified point on the Y path. For details, see pointY3DPath , on page 74.
<code>func.pointZ3DPath("columnName", point);</code>	Returns the value of the specified point on the Z path. For details, see pointZ3DPath , on page 74.
Editing Functions	
<code>func.addCtrlPointAfter3DPath("columnName", point, X, Y, Z, tension, continuity, bias);</code>	Adds a keyframe after a point on a 3D Path and sets the X, Y and Z values, as well as the tension, continuity and bias. For details, see addCtrlPointAfter3DPath , on page 68.
<code>func.addKeyFrame3DPath("columnName", frame, X, Y, Z, tension, continuity, bias);</code>	Adds a keyframe to a 3D Path and sets the X, Y and Z value, as well as the tension, continuity and bias. For details, see addKeyFrame3DPath , on page 68.
<code>func.removePoint3DPath("columnName", point);</code>	Removes either a control point or a keyframe from the 3D Path. For details, see removePoint3DPath , on page 74.
<code>func.setPoint3DPath("columnName", point, X, Y, Z, tension, continuity, bias);</code>	Sets the properties of a point on a 3D Path, including X, Y, and Z values, and tension, continuity and bias values. For details, see setPoint3DPath , on page 76.
<code>func.setBezierPoint("columnName", frame, Y, handleLeftX, handleLeftY, handleRightX, handleRightY, constSeg, "continuity");</code>	Sets the values of a point on a Bezier function curve. For details, see setBezierPoint , on page 75.
<code>func.setEasePoint("columnName", frame, Y, easeIn, angleEaseIn, easeOut, angleEaseOut, constSeg, "continuity");</code>	Sets the values of a point on an Ease function curve. For details, see setEasePoint , on page 75.

Function Curve Function	Description
<code>func.setHoldStartFrame("columnName", startFrame);</code>	Sets the Start value in the Hold Value Editor dialog box, for Bezier, Ease and Velo-based Function Editors. For details, see <code>setHoldStartFrame</code> , on page 76.
<code>func.setHoldStep("columnName", stepNumber);</code>	Sets the Hold value in the Hold Value Editor dialog box, for Bezier, Ease and Velo-based Function Editors. For details, see <code>setHoldStep</code> , on page 76.
<code>func.setHoldStopFrame("columnName", stopFrame);</code>	Sets the Stop value in the Hold Value Editor dialog box, for Bezier, Ease and Velo-based Function Editors. For details, see <code>setHoldStopFrame</code> , on page 76.
<code>func.setVeloBasedPoint("columnName", frame, Y);</code>	Sets the values of a point on a Velocity-Based function curve. For details, see <code>setVeloBasedPoint</code> , on page 77.

MessageLog Function	Description
<code>MessageLog.debug(message);</code>	Writes the message to the message log if debug mode is on. For details, see <code>debug</code> , on page 78.
<code>MessageLog.isDebugEnabled();</code>	Returns whether debug mode is enabled in the messageLog. For details, see <code>isDebugEnabled</code> , on page 78.
<code>MessageLog.setDebug(b);</code>	Sets the debug mode in message log. This will set debug mode for the entire application. For details, see <code>setDebug</code> , on page 78.
<code>MessageLog.trace(message);</code>	Writes the message to the message log. For details, see <code>trace</code> , on page 78.

Node Function	Description
<code>node.add("parent_group_path", "name", "type", X, Y, Z);</code>	Adds a module to the network. For details, see <code>add</code> , on page 81.
<code>node.addCompositeToGroup();</code>	Add a composite to the given group. For details, see <code>addCompositeToGroup</code> , on page 81.
<code>node.coordX("node_path");</code>	Returns an integer indicating the X position of a module in the network. For details, see <code>coordX</code> , on page 81.
<code>node.coordY("node_path");</code>	Returns an integer indicating the Y position of a module in the network. For details, see <code>coordY</code> , on page 81.
<code>node.createGroup();</code>	Create a group with the selected nodes. For details, see <code>createGroup</code> , on page 82.
<code>node.deleteNode();</code>	Delete a single node. Optionally, delete the columns and elements used by that node. For details, see <code>deleteNode</code> , on page 82.
<code>node.dstNode("node_path", iPort, iLink);</code>	Returns the path of the destination module linked to by the output port on the source module. For details, see <code>dstNode</code> , on page 82.
<code>node.equals("node_path", "node_path");</code>	Returns true or false to indicate if a node path is equal to another. For details, see <code>equals</code> , on page 82.
<code>node.explodeGroup();</code>	Explode a provided group. For details, see <code>explodeGroup</code> , on page 83.
<code>node.flatDstNode("node_path", iPort, iLink);</code>	If the <code>dstNode</code> is a Group Module, this returns the path of the module inside the Group Module that is the destination. For details, see <code>flatDstNode</code> , on page 83.
<code>node.flatSrcNode("node_path", iPort);</code>	If the <code>srcNode</code> is a Group Module, this returns the path of the module inside the Group Module that is the source. For details, see <code>flatSrcNode</code> , on page 83.
<code>node.getCameras();</code>	Returns list of all cameras. For details, see <code>getCameras</code> , on page 83.

Node Function	Description
<code>node.getDefaultCamera();</code>	Returns name of default camera. For details, see <code>getDefaultCamera</code> , on page 84.
<code>node.getEnable();</code>	Returns whether a module is enabled or not. For details, see <code>getEnable</code> , on page 84.
<code>node.getMatrix();</code>	Returns the model matrix for the node. For details, see <code>getMatrix</code> , on page 84.
<code>node.getName("node_path");</code>	Returns the name of a module. For details, see <code>getName</code> , on page 84.
<code>node.getTextAttr("node_path", atFrame, "attrName");</code>	Returns the value(s) of the module's selected attribute. For details, see <code>getTextAttr</code> , on page 84.
<code>node.isGroup("node_path");</code>	Returns a true or false value indicating if the module is a Group Module. For details, see <code>isGroup</code> , on page 85.
<code>node.isLinked("node_path", iPort);</code>	Returns true or false to indicate if a port is connected to another module. For details, see <code>isLinked</code> , on page 85.
<code>node.link("srcNode_path", srcPort, "dstNode_path", dstPort);</code>	Links a port on a module to a port on another module. For details, see <code>link</code> , on page 85.
<code>node.linkAttr("node_path", "attrName", "columnName");</code>	Links an attribute to a function column in the Xsheet View. For details, see <code>linkAttr</code> , on page 85.
<code>node.linkedColumn("node_path", "attrName");</code>	Returns the name of the column that an attribute is linked to. If the attribute is not linked to a column, the function returns the null string. For details, see <code>linkedColumn</code> , on page 86.
<code>node.noNode();</code>	Returns the null string that is returned by other functions when there is an error. For details, see <code>noNode</code> , on page 86.
<code>node.numberOfInputPorts("node_path ");</code>	Returns an integer indicating the number of input ports on the module. For details, see <code>numberOfInputPorts</code> , on page 86.
<code>node.numberOfOutputLinks("node_pat h", iPort);</code>	Returns an integer indicating the number of modules actually linked from the output ports. For details, see <code>numberOfOutputLinks</code> , on page 86.
<code>node.numberOfOutputPorts("node_pat h");</code>	Returns an integer indicating the number of output ports on a module. For details, see <code>numberOfOutputPorts</code> , on page 87.
<code>node.numberOfSubNodes("node_path") ;</code>	Returns an integer that indicates the number of modules contained in a group. For details, see <code>numberOfSubNodes</code> , on page 87.
<code>node.parentNode("node_path");</code>	Returns the path of the parent level of a module contained in a group. For details, see <code>parentNode</code> , on page 87.
<code>node.rename("node_path", "newName");</code>	Changes the name of a module. For details, see <code>rename</code> , on page 87.
<code>node.root();</code>	Returns the name of the Top level in the network, which is "Top". For details, see <code>root</code> , on page 88.
<code>node.setAsDefaultCamera("cam1");</code>	Sets the default camera. For details, see <code>setAsDefaultCamera</code> , on page 88.
<code>node.setAsGlobalDisplay();</code>	Select the provided display as the current global display. For details, see <code>setAsGlobalDisplay</code> , on page 88.
<code>node.setCoord("node_path", X, Y);</code>	Sets the position of a module in the network. For details, see <code>setCoord</code> , on page 88.
<code>node.setEnable(true);</code>	enables or disables a module For details, see <code>setEnable</code> , on page 88.
<code>node.setGlobalToDisplayAll();</code>	Select the "Display All" display as the current global display. For details, see <code>setGlobalToDisplayAll</code> , on page 89.

Node Function	Description
<code>node.setTextAttr("node_path", "attrName", atFrame, "attrValue");</code>	Changes the value of an attribute in a module. For details, see <code>setTextAttr</code> , on page 89.
<code>node.srcNode("node_path", iPort);</code>	Returns the path for the module that the port is linked to. For details, see <code>srcNode</code> , on page 89.
<code>node.subNode("node_path_parent", iSubNode);</code>	Returns the path of a module in a group. For details, see <code>subNode</code> , on page 89.
<code>node.subNodeByName ();</code>	Returns the path of a module in a group. For details, see <code>subNodeByName</code> , on page 90.
<code>node.type("node_path");</code>	Returns the module type. For details, see <code>type</code> , on page 90.
<code>node.unlink("dstNode_path", inPort);</code>	Unlinks a port on one module from the port on another module. For details, see <code>unlink</code> , on page 90.
<code>node.unlinkAttr("node_path", "attrName");</code>	Unlinks an attribute from a function column. For details, see <code>unlinkAttr</code> , on page 90.
<code>node.height ();</code>	Returns the node height. For details, see <code>width/height</code> , on page 91.
<code>node.width ();</code>	Returns the node width. For details, see <code>width/height</code> , on page 91.
PaletteManager Function	Description
<code>PaletteManager.getCurrentColorId ();</code>	This function retrieves the id of the currently selected colour.. For details, see <code>getCurrentColorId</code> , on page 92.
<code>PaletteManager.getCurrentColorName ();</code>	This function returns the current colour name from the <code>ColourView</code> . For details, see <code>getCurrentColorName</code> , on page 92.
<code>PaletteManager.getCurrentPaletteId ();</code>	This function returns the id of the current palette from the <code>ColourView</code> . For details, see <code>getCurrentPaletteld</code> , on page 92.
<code>PaletteManager.getCurrentPaletteName ();</code>	This function returns the current palette name from the <code>ColourView</code> . For details, see <code>getCurrentPaletteName</code> , on page 93.
<code>PaletteManager.setCurrentPaletteById ("palette");</code>	This function sets the current palette in the <code>ColourView</code> . For details, see <code>setCurrentPaletteById</code> , on page 93.
<code>PaletteManager.setCurrentColorById ("color");</code>	This function sets the current colour in the <code>ColourView</code> . For details, see <code>setCurrentColorById</code> , on page 93.
<code>PaletteManager.setCurrentPaletteAndColorById("palette", "color");</code>	This function sets the current palette and colour in the <code>ColourView</code> . For details, see <code>setCurrentPaletteAndColorById</code> , on page 93.
<code>PaletteManager.getCurrentPaletteSize ();</code>	This function returns the size of the currently selected palette in the <code>ColourView</code> . For details, see <code>getCurrentPaletteSize</code> , on page 93.
<code>PaletteManager.getColorName(int index);</code>	This function returns the name of the colour in the currently selected palette. For details, see <code>getColorName</code> , on page 94.
<code>PaletteManager.getColorId(int index);</code>	This function returns the id of the colour in the currently selected palette. For details, see <code>getColorId</code> , on page 94.
<code>PaletteManager.getNumPalettes ();</code>	This function returns the number of palettes in the current selected palette list in <code>ColourView</code> list. For details, see <code>getNumPalettes</code> , on page 94.
<code>PaletteManager.getNumPalettes (scenePaletteList);</code>	This function returns the number of palettes in palette list in <code>ColourView</code> . For details, see <code>getNumPalettes</code> , on page 94.

PaletteManager Function	Description
<code>PaletteManager.getPaletteName(int index);</code>	This function returns the name of the palette in the current palette list in the ColourView. For details, see <code>getPaletteName</code> , on page 94.
<code>PaletteManager.getPaletteName(int index, scenePaletteList)</code>	This function returns the name of the palette in the current palette list in the ColourView. For details, see <code>getPaletteName</code> , on page 95.
<code>PaletteManager.getPaletteId(int index);</code>	This function returns the id of the palette in the current palette list in the ColourView. For details, see <code>getPaletteld</code> , on page 95.
<code>PaletteManager.getPaletteId(int index, scenePaletteList);</code>	This function returns the id of the palette in the current palette list in the ColourView. For details, see <code>getPaletteld</code> , on page 95.

PenStyleManager Function	Description
<code>PenstyleManager.getNumberOfPenstyles();</code>	This function returns the number of penstyles. For details, see <code>getNumberOfPenstyles</code> , on page 97.
<code>PenstyleManager.getPenstyleName(int index);</code>	This function returns the name of the penstyle. For details, see <code>getPenstyleName</code> , on page 97.
<code>PenstyleManager.getCurrentPenstyleName();</code>	This function returns the name of the current pen style. For details, see <code>getCurrentPenstyleName</code> , on page 97.
<code>PenstyleManager.setCurrentPenstyleByName("name");</code>	This function sets the current penstyle by name. For details, see <code>setCurrentPenstyleByName</code> , on page 97.
<code>PenstyleManager.setCurrentPenstyleByIndex(int index);</code>	This function sets the current penstyle by index. For details, see <code>setCurrentPenstyleByIndex</code> , on page 97.
<code>PenstyleManager.changeCurrentPenstyleMinimumSize</code>	This function sets the current penstyle minimum size. For details, see <code>changeCurrentPenstyleMinimumSize</code> , on page 98.
<code>PenstyleManager.changeCurrentPenstyleMaximumSize(double maximum);</code>	This function sets the current penstyle maximum size. For details, see <code>changeCurrentPenstyleMaximumSize</code> , on page 98.
<code>PenstyleManager.getCurrentPenstyleIndex();</code>	This function sets the index of the current penstyle. For details, see <code>getCurrentPenstyleIndex</code> , on page 99.
<code>PenstyleManager.changeCurrentPenstyleOutlineSmoothness(int smooth);</code>	This function sets the current penstyle outline smoothness. For details, see <code>changeCurrentPenstyleOutlineSmoothness</code> , on page 98.
<code>PenstyleManager.changeCurrentPenstyleCenterlineSmoothness(int smooth);</code>	This function sets the current penstyle centreline smoothness. For details, see <code>changeCurrentPenstyleCenterlineSmoothness</code> , on page 98.
<code>PenstyleManager.changeCurrentPenstyleEraserFlag(flag);</code>	This function sets the current penstyle eraser flag. For details, see <code>changeCurrentPenstyleEraserFlag</code> , on page 98.
<code>PenstyleManager.getCurrentPenstyleMinimumSize();</code>	This function gets the current penstyle minimum size. For details, see <code>getCurrentPenstyleMinimumSize</code> , on page 99.
<code>PenstyleManager.getCurrentPenstyleMaximumSize();</code>	This function gets the current penstyle maximum size. For details, see <code>getCurrentPenstyleMaximumSize</code> , on page 99.
<code>PenstyleManager.getCurrentPenstyleOutlineSmoothness();</code>	This function gets the current penstyle outline smoothness. For details, see <code>getCurrentPenstyleOutlineSmoothness</code> , on page 99.
<code>PenstyleManager.getCurrentPenstyleEraserFlag();</code>	This function gets the current penstyle eraser flag. For details, see <code>getCurrentPenstyleEraserFlag</code> , on page 100.
<code>PenstyleManager.exportPenstyleToString(int index)</code>	This function creates a string representing the penstyle which can be used to store the penstyle and import it later. For details, see <code>exportPenstyleToString</code> , on page 100.

PenStyleManager Function	Description
<code>PenstyleManager.exportPenstyleListToString();</code>	This function formats the penstyle list into a string, which can be used to store the penstyle list and import it later. For details, see <code>exportPenstyleListToString</code> , on page 100.
<code>PenstyleManager.importPenstyleListFromString("str");</code>	This function imports a penstyle list from a previously formatted penstyle string. For details, see <code>importPenstyleListFromString</code> , on page 100.
<code>PenstyleManager.savePenstyles();</code>	This function saves the pen styles. For details, see <code>getNumberOfPenstyles</code> , on page 97.
<code>PenstyleManager.savePenstyles();</code>	This function saves the pen styles. For details, see <code>getNumberOfPenstyles</code> , on page 97.

Preferences Function	Description
<code>preferences.get();</code>	Retrieve the value of a boolean preference. For details, see <code>getBool</code> , on page 101.
<code>preferences.getColor();</code>	Retrieve the value of a colour preference. For details, see <code>getColor</code> , on page 102.
<code>preferences.getDouble();</code>	Retrieve the value of a double preference. For details, see <code>getDouble</code> , on page 102.
<code>preferences.getInt();</code>	Retrieve the value of a integer preference. For details, see <code>getInt</code> , on page 102.
<code>preferences.getString();</code>	Gets the value of a string preference. For details, see <code>getString</code> , on page 102.
<code>preferences.setBool();</code>	Set the value of a boolean preference. For details, see <code>setBool</code> , on page 102.
<code>preferences.setColor();</code>	Set the value of a colour preference. For details, see <code>setColor</code> , on page 103.
<code>preferences.setDouble();</code>	Set the value of a double preference. For details, see <code>setDouble</code> , on page 103.
<code>preferences.setInt();</code>	Set the value of a integer preference. For details, see <code>setInt</code> , on page 103.
<code>preferences.setString();</code>	Sets the value of a string preference. For details, see <code>setString</code> , on page 103.

Render Function	Description
<code>render.frameReady(int frame, SM_CelWrapper &frameCel);</code>	Event that notifies the script that a certain frame is available and at which location. For details, see <code>frameReady</code> , on page 105.
<code>render.renderFinished();</code>	Event that notifies the script when the render has completed. For details, see <code>renderFinished</code> , on page 105.
<code>render.setCombine(autoCombine, secondFieldFirst);</code>	Set if rendered frames sets should be combined and in which order. Specify these options if you are rendering in PAL or NTSC format. For details, see <code>setCombine</code> , on page 105.
<code>render.setFieldType(int type);</code>	Sets the frame output format. For details, see <code>setFieldType</code> , on page 105.
<code>render.setBgColor(QColor bgColor);</code>	Set the background colour to use when rendering in scene machine mode. For details, see <code>setBgColor</code> , on page 106.
<code>render.setResolution(int x, int y);</code>	Set the scene resolution to use for rendering. For details, see <code>setResolution</code> , on page 106.
<code>render.setRenderDisplay("name");</code>	Set which display module to use for rendering. "Display All" uses the global unconnected display module. For details, see <code>setRenderDisplay</code> , on page 106.

Render Function	Description
<code>render.setWriteEnabled(enabled);</code>	Enable or disable write modules during the render. For details, see <code>setWriteEnabled</code> , on page 106.
<code>render.renderScene(int fromFrame, int toFrame);</code>	Render a part of the scene. For details, see <code>renderScene</code> , on page 106.
<code>render.renderSceneAll();</code>	Render the complete scene. For details, see <code>renderSceneAll</code> , on page 107.
<code>render.cancelRender();</code>	Interrupt an active render. For details, see <code>cancelRender</code> , on page 107.

Scene Function	Description
<code>scene.beginUndoRedoAccum("commandname");</code>	Starts the accumulation of all of the functions between it and the <code>endUndoRedoAccum</code> function as one command that will appear in the undo/redo list. If you do not use this function with <code>endUndoRedoAccum</code> , each function in the script generates a separate undo/redo entry. For details, see <code>beginUndoRedoAccum</code> , on page 109.
<code>scene.cancelUndoRedoAccum();</code>	Cancel the accumulation of undo commands. Rollback all executed commands. For details, see <code>cancelUndoRedoAccum</code> , on page 109.
<code>scene.clearHistory</code>	This function clears the command history. After this call, it is not possible to undo the command. For details, see <code>clearHistory</code> , on page 109.
<code>scene.coordAtCenterX();</code>	Returns the X value of the centre coordinate of the scene grid. For details, see <code>coordAtCenterX</code> , on page 110.
<code>scene.coordAtCenterY();</code>	Returns the Y value of centre coordinate of the scene grid. For details, see <code>coordAtCenterY</code> , on page 110.
<code>scene.currentEnvironment();</code>	Returns the current environment name. For details, see <code>currentEnvironment</code> , on page 110.
<code>scene.currentJob();</code>	Returns the current job name. For details, see <code>currentJob</code> , on page 110.
<code>scene.currentProjectPath();</code>	Returns the current project path. For details, see <code>currentProjectPath</code> , on page 110.
<code>scene.currentProjectPathRemapped();</code>	Returns the current project path with shortcut translated and symlink resolved. For details, see <code>currentProjectPathRemapped</code> , on page 111.
<code>scene.currentResolutionX();</code>	Returns the preview resolution in X. For details, see <code>currentResolutionX</code> , on page 111.
<code>scene.currentResolutionY();</code>	Returns the previous resolution in Y. For details, see <code>currentResolutionY</code> , on page 111.
<code>scene.currentScene();</code>	Returns the name of the current scene. For details, see <code>currentScene</code> , on page 111.
<code>scene.currentVersion();</code>	Returns the current version of project/scene. For details, see <code>currentVersion</code> , on page 111.
<code>scene.defaultResolutionFOV();</code>	Returns the field of view. For details, see <code>defaultResolutionFOV</code> , on page 112.
<code>scene.defaultResolutionName();</code>	Returns the name of the scene resolution preset. For details, see <code>defaultResolutionName</code> , on page 112.
<code>scene.defaultResolutionX();</code>	Returns the scene resolution in X. For details, see <code>defaultResolutionX</code> , on page 112.
<code>scene.defaultResolutionY();</code>	Returns the scene resolution in Y. For details, see <code>defaultResolutionY</code> , on page 112.

Scene Function	Description
<code>scene.endUndoRedoAccum();</code>	Ends the accumulation all of the functions between it and the <code>beginUndoRedoAccum</code> function as one command that will appear in the undo/redo list. If you do not use this function with <code>beginUndoRedoAccum</code> , each function in the script generates a separate undo/redo entry. For details, see <code>endUndoRedoAccum</code> , on page 112.
<code>scene.fromOGL();</code>	Convert an OGL coordinate to a field coordinate. For details, see <code>fromOGL</code> , on page 113.
<code>scene.getCameraMatrix();</code>	Returns the model matrix for the default camera. Note that this is the inverse of the view matrix. For details, see <code>getCameraMatrix</code> , on page 113.
<code>scene.getFrameRate();</code>	Returns the frame rate. For details, see <code>getFrameRate</code> , on page 113.
<code>scene.numberOfUnitsX();</code>	Returns the number of units in the X-axis of the scene grid. For details, see <code>numberOfUnitsX</code> , on page 113.
<code>scene.numberOfUnitsY();</code>	Returns the number of units in the Y-axis of the scene grid. For details, see <code>numberOfUnitsY</code> , on page 113.
<code>scene.numberOfUnitsZ();</code>	Returns the number of units in the Z-axis of the scene grid. For details, see <code>numberOfUnitsZ</code> , on page 114.
<code>scene.saveAll();</code>	Save project, all drawings, palettes and palette list. For details, see <code>saveAll</code> , on page 114.
<code>scene.saveAsNewVersion("name.true");</code>	Save all, while creating a new version of project. For details, see <code>saveAsNewVersion</code> , on page 114.
<code>scene.setUnitsAspectRatio(x, y);</code>	Sets the aspect ratio of the scene. For details, see <code>setUnitsAspectRatio</code> , on page 115.
<code>scene.setNumberOfUnits(x, y, z);</code>	Sets the number of X, Y, and Z units in the scene grid. For details, see <code>setNumberOfUnits</code> , on page 114.
<code>scene.setCoordAtCenter(x, y);</code>	Sets the value of the centre (X, Y) coordinates. For details, see <code>setCoordAtCenter</code> , on page 114.
<code>scene.setDefaultResolution();</code>	Set the default resolution X,Y and field of view. For details, see <code>setDefaultResolution</code> , on page 115.
<code>scene.setFrameRate();</code>	Set the global frame rate. For details, see <code>setFrameRate</code> , on page 116.
<code>scene.toOGL();</code>	Converts a field coordinate to an OGL coordinate. For details, see <code>toOGL</code> , on page 115.
<code>scene.unitsAspectRatioX();</code>	Returns the X value of the aspect ratio of the cells in the scene grid. For details, see <code>unitsAspectRatioX</code> , on page 115.
<code>scene.unitsAspectRatioY();</code>	Returns the Y value of the aspect ratio of the cells in the scene grid. For details, see <code>unitsAspectRatioY</code> , on page 115.
Selection Function	Description
<code>selection.clearSelection();</code>	This function clears the selection. For details, see <code>clearSelection</code> , on page 118.
<code>selection.addDrawingColumnToSelection(columnName);</code>	This function adds the drawing column and it's associated read node to the selection. For details, see <code>addDrawingColumnToSelection</code> , on page 118
<code>selection.addColumnToSelection(column);</code>	This function adds a column to the selection. For details, see <code>addColumnToSelection</code> , on page 118.
<code>selection.addNodeToSelection(node);</code>	This function adds a node to the selection. For details, see <code>addNodeToSelection</code> , on page 118.

Selection Function	Description
<code>selection.extendSelectionWithColumn(columnName);</code>	This function adds the drawing column to the selection. If the column is a drawing column, also adds the associated read node to the selection. For details, see <code>extendSelectionWithColumn</code> , on page 118.
<code>selection.numberOfCellColumnsSelected();</code>	This function returns a value for the number of selected columns. For details, see <code>numberOfCellColumnsSelected</code> , on page 119.
<code>selection.numberOfFramesSelected();</code>	This function returns the number of frames -selected (to be used in the <code>hxheetview</code> only). For details, see <code>numberOfFramesSelected</code> , on page 119.
<code>selection.numberOfNodesSelected();</code>	This function returns the number of modules that are selected. For details, see <code>numberOfNodesSelected</code> , on page 119.
<code>selection.selectAll();</code>	This function selects all nodes and all columns in the scene. For details, see <code>selectAll</code> , on page 119.
<code>selection.selectedCellColumn(int i);</code>	This function returns the name of the selected column. For details, see <code>selectedCellColumn</code> , on page 119.
<code>selection.selectedNode(int i);</code>	This function returns the path of the selected node. For details, see <code>selectedNode</code> , on page 120.
<code>selection.setSelectionFrameRange(int start, int end);</code>	This function sets the frame range for the selection. For details, see <code>setSelectionFrameRange</code> , on page 120.
Sound Function	Description
<code>sound.setSampleRate(double rate);</code>	This function sets the audio sample rate. For details, see <code>setSampleRate</code> , on page 121.
<code>sound.setChannelSize(int size);</code>	Sets the audio channel size (i.e. 8 or 16 bit). For details, see <code>setChannelSize</code> , on page 121.
<code>sound.setChannelCount(int count);</code>	Sets the number of audio channels (i.e 1 for mono and 2 for stereo). For details, see <code>setChannelCount</code> , on page 121.
<code>sound.getSoundtrack(int fromFrame, int toFrame);</code>	Returns a part of the scene's soundtrack in a temporary file in WAV format. For details, see <code>getSoundtrack</code> , on page 122.
<code>sound.getSoundtrackAll();</code>	Returns the scene's soundtrack in a temporary file in WAV format. For details, see <code>getSoundtrackAll</code> , on page 122.
SpecialFolders Function	Description
<code>specialFolders.app;</code>	Returns the global application folder. For details, see <code>app</code> , on page 123.
<code>specialFolders.bin;</code>	Returns the global binary folder. For details, see <code>bin</code> , on page 124.
<code>specialFolders.config;</code>	Returns the global configuration file folder. For details, see <code>config</code> , on page 124.
<code>specialFolders.etc;</code>	Returns the global./etc folder. For details, see <code>etc</code> , on page 124.
<code>specialFolders.htmlHelp;</code>	Returns the path to html folder. For details, see <code>htmlHelp</code> , on page 124.
<code>specialFolders.lang;</code>	Returns the language folder. For details, see <code>lang</code> , on page 124.
<code>specialFolders.library;</code>	Returns the platform library folder. For details, see <code>library</code> , on page 125.
<code>specialFolders.pdf</code>	Returns the path to pdf folder. For details, see <code>platform</code> , on page 125.
<code>specialFolders.platform;</code>	Returns the platform folder. For details, see <code>platform</code> , on page 125.
<code>specialFolders.plugins;</code>	Returns the platform plug-in folder. For details, see <code>plugins</code> , on page 125.

SpecialFolders Function	Description
<code>specialFolders.resource;</code>	Returns the resource folder. For details, see resource, on page 125.
<code>specialFolders.root;</code>	Returns the installation root folder. For details, see root, on page 126.
<code>specialFolders.temp;</code>	Returns the platform temporary folder. For details, see temp, on page 126.
<code>specialFolders.userConfig;</code>	Returns the user configuration file folder. For details, see userConfig, on page 126.

Timeline Function	Description
<code>Timeline.firstFrameSel</code>	Returns the number of the first frame in the Timeline selection or the current frame, if only one frame is selected. For details, see firstFrameSel, on page 129.
<code>Timeline.isAncestorOf (parentLayerIdx, layerIdx);</code>	Returns true or false to identify if a layer is the parent of another layer. For details, see isAncestorOf, on page 129.
<code>Timeline.layerIsColumn (layerIdx);</code>	Returns true or false to identify if the Timeline layer is linked to a column in the Xsheet. For details, see layerIsColumn, on page 129.
<code>Timeline.layerIsNode (layerIdx);</code>	Returns true or false to identify if the Timeline layer is linked to a module (node) in the Network. For details, see layerIsNode, on page 129.
<code>Timeline.layerToColumn (layerIdx);</code>	Returns the column name for the Timeline layer. It returns an empty string if the layer is not a column. For details, see layerToColumn, on page 130.
<code>Timeline.layerToNode (layerIdx);</code>	Returns the node (module) index from the Network for the Timeline layer. It returns an empty string if the layer is not a node. For details, see layerToNode, on page 130.
<code>Timeline.numFrameSel</code>	Returns the number of the selected frame in the Timeline, if only one frame is selected. For details, see numFrameSel, on page 130.
<code>Timeline.numLayers</code>	Returns the number of layers in the Timeline. For details, see numLayers, on page 130.
<code>Timeline.numLayersSel</code>	Returns the number of layers that are selected in the Timeline. For details, see numLayerSel, on page 130.
<code>Timeline.parentNodeIndex (layerIdx)</code>	Returns a layer identifier (layerIdx) for the parent of the layer (layerIdx). For details, see parentNodeIndex, on page 131.
<code>Timeline.selIsColumn (selIdx)</code>	Returns true or false to indicate if the Timeline selection has a column in the Xsheet. For details, see selIsColumn, on page 131.
<code>Timeline.selIsNode (selIdx)</code>	Returns true or false to identify if the Timeline selection is linked to a module (node) in the Network. For details, see selIsNode, on page 131.
<code>Timeline.selToColumn (selIdx)</code>	Returns the column name for the Timeline selection. For details, see selToColumn, on page 131.
<code>Timeline.selToLayer (selIdx)</code>	Returns the layer identifier (layerIdx) for the selection (selIdx) in the Timeline. For details, see selToLayer, on page 132.
<code>Timeline.selToNode (selIdx)</code>	Returns the node name from the module in the Network for the Timeline selection. For details, see selToNode, on page 132.
<code>Timeline.setDisplayToUnconnected ();</code>	Make the "Display All" display the current display. This method is obsolete. For details, see setDisplayToUnconnected, on page 132.

View Function	Description
<code>view.column(currentView);</code>	Returns the name of the column for the currently displayed function in the Function View. For details, see column , on page 133.
<code>view.currentView();</code>	Returns a unique identifier for the current, active View. For details, see currentView , on page 133.
<code>view.group(currentView);</code>	Returns the name of the current Group Module in the active Network View. For details, see group , on page 133.
<code>view.refreshViews();</code>	Forces a redisplay of drawing and scene planning views. For details, see refreshViews , on page 134.
<code>view.type(currentView);</code>	Returns a string that indicates what type of View the <code>currentView</code> is. For details, see type , on page 134.

About

This set of functions returns information about the current application running the script and its environment.

Following are the About functions:

- `animate`, on page 37
- `animatePro`, on page 37
- `applicationPath`, on page 37
- `controlCenterApp`, on page 37
- `demoVersion`, on page 37
- `educVersion`, on page 38
- `fullVersion`, on page 38
- `getApplicationPath`, on page 38
- `getFlavorString`, on page 38
- `getVersionInfoStr`, on page 38
- `harmony`, on page 39
- `interactiveApp`, on page 39
- `isAnimate`, on page 39
- `isAnimatePro`, on page 39
- `isControlCenterApp`, on page 39
- `isDemoVersion`, on page 40
- `isEducVersion`, on page 40
- `isFullVersion`, on page 40
- `isHarmony`, on page 40
- `isInteractiveApp`, on page 40
- `isLinuxArch`, on page 41
- `isMacArch`, on page 41
- `isMacIntelArch`, on page 41
- `isMacPpcArch`, on page 41
- `isMainApp`, on page 41
- `isPaintMode`, on page 42
- `isScanApp`, on page 42
- `isStage()`, on page 42
- `isWindowsArch`, on page 42
- `isXsheetMode`, on page 42
- `linuxArch`, on page 43
- `macArch`, on page 43
- `macIntelArch`, on page 43
- `macPpcArch`, on page 43
- `mainApp`, on page 43
- `paintMode`, on page 44
- `productName`, on page 44
- `scanApp`, on page 44
- `stage`, on page 44
- `windowsArch`, on page 44
- `xsheetMode`, on page 45

Example

```
function printAbout()
{
    // Application : normal, demo or educational
    print("");
    print( "full (commercial) version: " + Application.about.fullVersion );
    print( "demo version: " + Application.about.demoVersion );
    print( "educational version: " + Application.about.educVersion );
    print("");

    // Software / product
    print( "animate pro product : " + Application.about.animatePro );
    print( "harmony product : " + Application.about.harmony );
    print( "" );
    print( "stage: " + Application.about.stage );

    // Architecture
    print( "Windows architecture: " + Application.about.windowsArch );
    print( "OSX architecture: " + Application.about.macArch );
    print( "OSX PowerPC architecture: " + Application.about.macPpcArch );
    print( "OSX Intel architecture: " + Application.about.macIntelArch );
    print( "Linux architecture: " + Application.about.linuxArch );

    // type of application
    print( "Interactive: " + Application.about.interactiveApp );

    print( "Stage/Digital Main Mode: " + Application.about.mainApp );
    print( "Paint mode: " + Application.about.paintMode );
    print( "XSheet mode: " + Application.about.xsheetMode );

    print( "Scan application: " + Application.about.scanApp );
    print( "ControlCenter application: " + Application.about.controlCenterApp);

    // application path...
    print( "application path: " + Application.about.applicationPath );

    print("");
}
}
```

animate

Description

This is a read-only property that is true whenever this script is being executed by Animate.

Syntax

```
about.animate
```

Arguments

None

animatePro

Description

This is a read-only property that is true whenever this script is being executed by Animate Pro.

Syntax

```
about.animatePro
```

Arguments

None

applicationPath

Description

This is a read-only property that contains the application path of the currently executing module.

Syntax

```
about.applicationPath
```

Arguments

None

controlCenterApp

Description

This is a read-only property that is true whenever this script is being executed by ControlCenter.

Syntax

```
about.controlCenterApp
```

Arguments

None

demoVersion

Description

This is a read-only property that is true whenever this script is being executed by a demo version.

Syntax

```
about.demoVersion
```

Arguments

None

educVersion

Description

This is a read-only property that is true whenever this script is being executed in a Education version.

Syntax

```
about.educVersion
```

Arguments

None

fullVersion

Description

This is a read-only property that is true whenever this script is being executed in a Commercial Standard version.

Syntax

```
about.fullVersion
```

Arguments

None

getApplicationPath

Description

This is a function that returns the application. Identical to property "applicationPath".

Syntax

```
about.getApplicationPath();
```

Arguments

None

getFlavorString

Description

This function returns a string that represents the flavour of the application, e.g. Harmony, Animate, Opus.

Syntax

```
about.getFlavorString();
```

Arguments

None

getVersionInfoStr

Description

This function returns the version info string.

Syntax

```
about.getVersionInfoStr();
```

Arguments

None

harmony

Description

This is a read-only property that is true whenever this script is being executed by Harmony.

Syntax

```
about.harmony
```

Arguments

None

interactiveApp

Description

This is a read-only property that is true whenever this application the running application is interactive. Currently, all application running scripts are interactive.

Syntax

```
about.interactiveApp
```

Arguments

None

isAnimate

Description

This is a read-only property that is true whenever this script is being executed by Animate.

Syntax

```
about.isanimate
```

Arguments

None

isAnimatePro

Description

This is a read-only property that is true whenever this script is being executed by Animate Pro.

Syntax

```
about.isanimatePro
```

Arguments

None

isControlCenterApp

Description

This function returns true whenever this application running application is ControlCenter

Syntax

```
about.isControlCenterApp()
```

Arguments

None

isDemoVersion

Description

This function returns true whenever this application is a Demo variant.

Syntax

```
about.isDemoVersion();
```

Arguments

None

isEducVersion

Description

This function returns true whenever this application is an Educational variant.

Syntax

```
about.isEducVersion();
```

Arguments

None

isFullVersion

Description

This function returns true whenever this application is a Commercial/Full variant.

Syntax

```
about.isFullVersion();
```

Arguments

None

isHarmony

Description

This is a read-only property that is true whenever this script is being executed by Harmony.

Syntax

```
about.isHarmony();
```

Arguments

None

isInteractiveApp

Description

This function returns true whenever this application is interactive. All application capable of running scripts are interactive.

Syntax

```
about.isInteractiveApp();
```

Arguments

None

isLinuxArch

Description

This function returns true when running on a Linux operating system.

Syntax

```
about.isLinuxArch();
```

Arguments

None

isMacArch

Description

This function returns true when running on an Mac OS X operating system.

Syntax

```
about.isMacArch();
```

Arguments

None

isMacIntelArch

Description

This function returns true when running on an Mac OS X operating system and on a Mac Intel.

Syntax

```
about.isMacIntelArch();
```

Arguments

None

isMacPpcArch

Description

This function returns true when running on an Mac OS X operating system and on a Mac PowerPC.

Syntax

```
about.isMacPpcArch();
```

Arguments

None

isMainApp

Description

This function returns true when this application is Stage, Digital Pro or Storyboard, and not a peripheral application.

Syntax

```
about.isMainApp ();
```

Arguments

None

isPaintMode

Description

This function returns true when this application is in Paint mode.

Syntax

```
about.isPaintMode();
```

Arguments

None

isScanApp

Description

This function returns true when this application is TbScan().

Syntax

```
about.isScanApp();
```

Arguments

None

isStage()

Description

This function returns true whenever the script is being executed by Stage.

Syntax

```
about.isStage();
```

Arguments

None

isWindowsArch

Description

This function returns true when running on Windows.

Syntax

```
about.isWindowsArch();
```

Arguments

None

isXsheetMode

Description

This function returns true when running Stage in Xsheet mode.

Syntax

```
about.isXsheetMode();
```

Arguments

None

linuxArch

Description

A read-only property that is true when running on Linux

Syntax

```
about.linuxArch;
```

Arguments

None

macArch

Description

A read-only property that is true when running on a Mac OS X operating system.

Syntax

```
about.macArch;
```

Arguments

None

macIntelArch

Description

A read-only property that is true when running on a Mac OS X operating system and on a Mac Intel.

Syntax

```
about.macIntelArch
```

Arguments

None

macPpcArch

Description

A read-only property that is true when running on a Mac OS X operating system and on a Mac PowerPC.

Syntax

```
about.macPpcArch ;
```

Arguments

None

mainApp

Description

A read-only property that is true when this application is Stage, Digital Pro or Storyboard, and not a peripheral application.

Syntax

```
about.mainApp;
```

Arguments

None

paintMode

Description

A read-only property that is true when this application is in Paint mode.

Syntax

```
about.paintMode ;
```

Arguments

None

productName

Description

This function returns a string that is the name of application.

Syntax

```
about.productName () ;
```

Arguments

None

scanApp

Description

A read-only property that is true when running TbScan.

Syntax

```
about.scanApp
```

Arguments

None

stage

Description

A read-only property that is true whenever the script is being executed by Stage.

Syntax

```
about.stage
```

Arguments

None

windowsArch

Description

A read-only property that is true when running on Windows.

Syntax

```
about.windowsArch
```

Arguments

None

xsheetMode

Description

A read-only property that is true when running Stage in Xsheet mode.

Syntax

```
about.xsheetMode.
```

Arguments

None

Action

This interface is used to perform menu or tool bar functions.

The following are the action functions:

- `perform`, on page 46

Example

```
function callAbout()  
{  
    Action.perform("onActionAbout()");  
}
```

perform

Description

This function calls the action it has to perform.

Syntax

```
Action.perform("onActionAbout()");
```

Arguments

- **name**: The name of the menu function.

Column

With the Column function, you can retrieve values from columns in your scene and you can add and remove them.

The following are the Column functions:

- add, on page 48
- clearKeyFrame, on page 48
- getColorForXSheet, on page 48
- getColumnListOfType, on page 48
- getCurrentVersionFor Drawing, on page 49
- getDisplayName, on page 49
- getDrawingName, on page 49
- getDrawingTimings, on page 49
- getElementIdOfDrawing, on page 50
- getEntry, on page 50
- getName, on page 50
- getNextKeyDrawing, on page 50
- getTextOfExpr, on page 51
- importSound, on page 51
- isKeyFrame, on page 51
- numberOf, on page 51
- rename, on page 52
- setColorForXSheet, on page 52
- setElementIdOfDrawing, on page 52
- setEntry, on page 53
- setKeyFrame, on page 53
- setTextOfExpr, on page 53
- type, on page 54

Example

In this script, a loop is run on each column to retrieve the name and type of each column. If a column is a Drawing column, the element is also printed retrieved. All this information is printed to the shell that started Toon Boom Animate.

```
function printColumnInfo()
{
    n = column.numberOf();
    System.println("Columns");
    for (i = 0; i < n; ++i)
    {
        var line;
        name = column.getDisplayName(i);
        type = column.type(name);
        line = i + " " + name + "(" + type + ")";
        if (type == "DRAWING")
            line += " element = "
            + column.getElementIdOfDrawing(name);
        System.println(line);
    }
}
```

add

Description

This function adds a column with the specified name and type.

Syntax

```
column.add("columnName", "columnType");
```

Arguments

"columnName", "columnType"

- **columnName**: The name of the column you want to add.
- **columnType**: The type of column you want to add. You can add any of the following: DRAWING, SOUND, 3DPATH, BEZIER, EASE, EXPR (for expression), TIMING, QUATERNION and ANNOTATION.

clearKeyFrame

Description

This function removes a keyframe from a cell in a column.

Syntax

```
column.clearKeyFrame("columnName", atFrame);
```

Arguments

"columnName", atFrame

- **columnName**: The name of the column. If the column is a 3D Path with multiple columns, the keyframe in each column will be removed.
- **atFrame**: The frame number where you want to clear the keyframe.

getColorForXSheet

Description

This function returns the colour for the given column.Syntax

Syntax

```
column.getColorForXSheet();
```

Arguments

- **columnName**: The name of the column.

getColumnListOfType

Description

This function returns a list of column names which match the given column type

Syntax

```
column.getColumnListOfType(type);
```

Arguments

- **type**: is the type string,i.e."DRAWING".

getCurrentVersionFor Drawing

Description

This function returns the current drawing version.

Syntax

```
column.getCurrentVersionForDrawing( columnName, timingName );
```

Arguments

- **columnName** – this is the string name of the column.
- **TimingName** – this is the string name of the drawing.

getDisplayname

Description

This function returns the displayable name (similar to the one displayed in the xsheet view) of a column.

Syntax

```
column.getDisplayName( "columnName" );
```

Arguments

- **columnName**: The name of the column.

getDrawingColumnList

Description:

This function returns a list of all drawing columns in the scene.

Syntax:

```
column.getDrawingColumnList();
```

Arguments:

None

getDrawingName

Description

This function returns the drawing name for the specified column at the specified frameSyntax

Syntax

```
column.getDrawingName( columnName, frame );
```

Arguments

- **columnName**: The name (unique identifier) of the column.
- **frame**: The frame number that you want to retrieve the value from

getDrawingTimings

Description

This function returns a list of all drawing names used in the drawing column.

Syntax

```
column.getDrawingTimings( columnName );
```

Arguments

columnName – this is the string name of the column.

getElementIdOfDrawing

Description

This function returns the id of the element linked to the specified drawing column.

Syntax

```
column.getElementIdOfDrawing("columnName");
```

Arguments

`"columnName"`

- `columnName`: The name of the column.

getEntry

Description

This function returns the value of a cell in a column.

Syntax

```
column.getEntry("columnName", subColumn, atFrame);
```

Arguments

`"columnName", subColumn, atFrame`

- `columnName`: The name of the column, enclosed in quotes
- `subColumn`: The number value of the sub-column. This only exists in the case of 3D Path columns, which include a group of sub-columns for the X, Y, Z and velocity values on the 3D Path. Each sub-column has a number:
 - ⇒ X = 1
 - ⇒ Y = 2
 - ⇒ Z = 3
 - ⇒ Velocity = 4
- `atFrame`: The frame number that you want to retrieve the value from.

getName

Description

This functions returns the unique identifier that names the column. This is not the display name shown in the xsheet view.

Syntax

```
column.getName(columnNumber);
```

Arguments

`columnNumber`

- `columnNumber`: This is an integer that represents the numerical value of the column. This integer is between 0 and `column.numberOf`.

getNextKeyDrawing

Description

This function returns the next key drawing in a drawing column.

Syntax

```
column.getNextKeyDrawing(columnName, startFrame );
```

Arguments

- `columnName`: This is the string that represents the columns name.
- `startFrame`: This is the frame number that specifies the search start point.

getTextOfExpr

Description

This function returns the expression text in the identified column.

Syntax

```
column.getTextOfExpr("columnName");
```

Arguments

"columnName"

- **columnName**: The name of the Expression column that contains the text you want to return. The columnName is must be enclosed in quotes.

importSound

Description

Import a sound file in the specified column at the specified frame. This function returns a Boolean indicating the success of the operation.

Syntax

```
column.importSound( columnName, atFrame, soundFilePath );
```

Arguments

- **columnName**: The column name identifying a sound column.
- **atFrame**: The frame number that specifies where the sound will be inserted in the timing.

isKeyFrame

Description

This function returns true or false indicating if a cell in a column is a keyframe.

Syntax

```
column.isKeyFrame("columnName", subColumn, atFrame);
```

Arguments

"columnName", subColumn, atFrame

- **columnName**: The name of the column.
- **subColumn**: The number value of the sub-column. This only exists in the case of 3D Path columns, which include a group of sub-columns for the X, Y, Z and velocity values on the 3D Path. Each sub-column has a number:
 - ⇒ X = 1
 - ⇒ Y = 2
 - ⇒ Z = 3
 - ⇒ Velocity = 4
- **atFrame**: The frame number that you want to retrieve the value from.

numberOf

Description

This function returns the number of columns in the scene.

Syntax

```
column.numberOf();
```

Arguments

None.

rename

Description

This function renames the specified column.

Syntax

```
column.rename( "oldName", "newName" );
```

Arguments

"oldName", "newName"

- **oldName**: The original name of the column you want to change.
- **newName**: The new name of the column.

setColorForXSheet

Description

This function sets the colour for an Xsheet column.

Syntax

```
column.setColorForXSheet( columnName, colour );
```

Arguments

- **columnName**: The name of the column.
- **colour**: A colour object of type Color.

Example

```
// create a pure RED colour object and sets this colour to the column name  
"Drawing"  
  
var c = new Color( 255,0,0 );  
  
column.setColorForXSheet( "Drawing", c );
```

setElementIdOfDrawing

Description

This function links an empty Drawing column to an element.

Syntax

```
column.setElementIdOfDrawing( "columnName", "elementId" );
```

Arguments

"columnName", "elementName"

- **columnName**: The name of the column.
- **elementId**: The id of the element you want to link to the column.

setEntry

Description

This function sets the value of a cell in a column.

Syntax

```
column.setEntry("columnName", subColumn, atFrame, "value");
```

Arguments

"columnName", subColumn, atFrame, "value"

- **columnName**: The name of the column, enclosed in quotes
- **subColumn**: The number value of the sub-column. This only exists in the case of 3D Path columns, which include a group of sub-columns for the X, Y, Z and velocity values on the 3D Path. Each sub-column has a number:
 - ⇒ X = 1
 - ⇒ Y = 2
 - ⇒ Z = 3
 - ⇒ Velocity = 4
- **atFrame**: The frame number.
- **value**: The value you want to set the cell to.

setKeyFrame

Description

This function makes a cell in a column a keyframe.

Syntax

```
column.setKeyFrame("columnName", atFrame);
```

Arguments

"columnName", atFrame

- **columnName**: The name of the column. If the column is a 3D Path with multiple columns, the frame in each column will be made a keyframe.
- **atFrame**: The frame number where you want to set the keyframe.

setTextOfExpr

Description

This function sets the value in the Expression column to the specified text.

Syntax

```
column.setTextOfExpr("columnName", "text");
```

Arguments

"columnName", "text"

- **columnName**: The name of the column.
- **text**: The expression text.

type

Description

This function returns the column type. There are nine column types: drawing (DRAWING), sound (SOUND), 3D Path (3DPATH), Bezier Curve (BEZIER), Ease Curve (EASE), Expression (EXPR), Timing (TIMING) for timing columns, Quaternion path (QUATERNIONPATH) for 3D rotation and Annotation (ANNOTATION) for annotation columns.

Syntax

```
column.type("columnName");
```

Arguments

`"columnName"`

- `columnName`: A string for the name of the column.

CopyPaste

The following are the CopyPaste functions:

- `createTemplateFromSelection`, on page 55
- `pasteTemplateIntoScene`, on page 55
- `usePasteSpecial`, on page 56
- `setPasteSpecialCreateNewColumn`, on page 56
- `setPasteSpecialElementTimingColumnMode`, on page 57
- `setPasteSpecialAddRemoveMotionKeyFrame`, on page 57
- `setPasteSpecialAddRemoveVelocityKeyFrame`, on page 57
- `setPasteSpecialAddRemoveAngleKeyFrame`, on page 57
- `setPasteSpecialAddRemoveSkewKeyFrame`, on page 57
- `setPasteSpecialAddRemoveScalingKeyFrame`, on page 58
- `setPasteSpecialForcesKeyFrameAtBegAndEnd`, on page 58
- `setPasteSpecialOffsetKeyFrames`, on page 58
- `setPasteSpecialReplaceExpressionColumns`, on page 58
- `setPasteSpecialDrawingAction`, on page 58
- `setPasteSpecialDrawingFileMode`, on page 59
- `setPasteSpecialColorPaletteOption`, on page 59

createTemplateFromSelection

Description

This function creates template from the current selection in the scene, using only the current drawing versions.

Syntax

```
copyPaste.createTemplateFromSelection("name", "path" );
```

Arguments

- **name**: This is the name of new template
- **path**: = This is the location of new template

pasteTemplateIntoScene

Description

This function pastes the template into the scene.

Syntax

```
copyPaste.pasteTemplateIntoScene( "templateSrcPath", insertColumnName,
insertFrame );
```

Arguments

- **templateSrcPath**: - This is the path of the template
- **insertColumnName**: - This is the name of existing column in which we will insert template
- **insertFrame**: - This is the frame at which insert commences

usePasteSpecial

Description

This function enable PasteSpecial. This is a STATIC structure - once initialized it may be re-used for the duration of the script. By default, it is NOT used until `usePasteSpecial(true)` is called.

Syntax

```
copyPaste.usePasteSpecial( flag );
```

Arguments

- `flag`: enables PasteSpecial mode

The PasteSpecial defaults are:

- `extendScene`: = false

MODULE

- `createNewColumn`: = false
- `elementTimingColumnMode`: = ELEMENT_AS_ELEMENT_AND_TIMING_AS_TIMING

PEG

- `add/remove motion KF`: = true
- `add/remove velocity KF`: = true
- `add/remove angle KF`: = true
- `add/remove skew KF`: = true
- `add/remove scaling KF`: = true
- `force keyframes at functions beginning and end`: = true
- `offset keyframes`: = false;
- `replace expression columns`: = true

DRAWING

- `drawing action` = add or remove exposure

PALETTE

- `palette mode` = REUSE_PALETTES

If Drawing add/remove exposure is set, the following modes are available

- ⇒ `drawing file mode` = ALWAYS_CREATE if the preference LIBRARY_PASTE_CREATE_NEW_DRAWING is set otherwise it is ONLY_CREATE_IF_DOES_NOT_EXIST
- ⇒ `automatically extend exposure` = false
- ⇒ `drawing as key frame` = true (note, only used if automatic extend exposure = true)

setPasteSpecialCreateNewColumn

Description

This function senables the CreateNewColumn mode for calls to `pasteTemplateIntoScene`.

Syntax

```
copyPaste.setPasteSpecialCreateNewColumn(flag );
```

Arguments

- `flag`: enables the CreateNewColumn mode.

setPasteSpecialElementTimingColumnMode

Description

This function sets the paste special element timing mode for calls to `pasteTemplateIntoScene`

Syntax

```
copyPaste.setPasteSpecialElementTimingColumnMode ( mode );
```

Acceptable strings are:

```
ELEMENT_AS_ELEMENT_AND_TIMING_AS_TIMING
```

```
ALL_DRWGS_AS_ELEMENTS
```

```
ALL_DRWGS_LINKED_THRU_TIMING_COLS
```

Arguments

- `mode`: string indicating element timing mode.

setPasteSpecialAddRemoveMotionKeyFrame

Description

PEGS: This function is a paste special option to add or remove motion key frames. See the paste special dialog.

Syntax

```
copyPaste.setPasteSpecialAddRemoveMotionKeyFrame(flag );
```

Arguments

- `flag`: enables the add/remove motion keyframes mode.

setPasteSpecialAddRemoveVelocityKeyFrame

Description

PEGS: This function is a paste special option to add or remove velocity key frames. See the paste special dialog.

Syntax

```
copyPaste.setPasteSpecialAddRemoveVelocityKeyFrame(flag );
```

Arguments

- `flag`: enables the add/remove velocity keyframes.

setPasteSpecialAddRemoveAngleKeyFrame

Description

PEGS: This function is a paste special option to add or remove angle key frames. See the paste special dialog.

Syntax

```
copyPaste.setPasteSpecialAddRemoveAngleKeyFrame(flag );
```

Arguments

- `flag`: enables the add/remove angle keyframe mode.

setPasteSpecialAddRemoveSkewKeyFrame

Description

PEGS: This function is a paste special option to add or remove skew key frames. See the paste special dialog.

Syntax

```
copyPaste.setPasteSpecialAddRemoveSkewKeyFrame(flag );
```

Arguments

- `flag`: enables the add/remove skew keyframe mode.

setPasteSpecialAddRemoveScalingKeyFrame

Description

PEGS: This function is a paste special option to add or remove scaling key frames. See the paste special dialog.

Syntax

```
copyPaste.setPasteSpecialAddRemoveScalingKeyFrame( flag );
```

Arguments

- **flag**: enables the add/remove scaling keyframe mode.

setPasteSpecialForcesKeyFrameAtBegAndEnd

Description

PEGS: This function is a paste special option to force key frames at the beginning and end of function columns. See the paste special dialog.

Syntax

```
copyPaste.setPasteSpecialForcesKeyFrameAtBegAndEnd( flag );
```

Arguments

- **flag**: enables the force keyframe mode.

setPasteSpecialOffsetKeyFrames

Description

PEGS: This function is a paste special option to offset keyframes in function columns. See the paste special dialog.

Syntax

```
copyPaste.setPasteSpecialOffsetKeyFrames( flag );
```

Arguments

- **flag**: enables the offset keyframe mode.

setPasteSpecialReplaceExpressionColumns

Description

PEGS: This function is a paste special option to replace the content of expression columns. See the paste special dialog.

Syntax

```
copyPaste.setPasteSpecialReplaceExpressionColumns( flag );
```

Arguments

- **flag**: enables the replace expression mode.

setPasteSpecialDrawingAction

Description

DRAWING: This function sets the drawing action mode. See the paste special dialog.

Acceptable Strings are:

- DO_NOTHING
- ADD_REMOVE_EXPOSURE
- UPDATE_PIVOT

Syntax

```
copyPaste.setPasteSpecialDrawingAction( "mode" );
```

Arguments

- **mode**: sets the drawing action mode.

setPasteSpecialDrawingFileMode

Description

DRAWING: This function sets the drawing file mode - only used if the DrawingAction is set to ADD_OR_REMOVE_EXPOSURE

Acceptable Strings are:

- NEVER_CREATE
- ONLY_CREATE_IF_DOES_NOT_EXIST
- ALWAYS_CREATE
- ALWAYS_CREATE_AND_VERSION_IF_NECESSARY

Syntax

```
copyPaste.setPasteSpecialDrawingFileMode
```

Arguments

- **mode**: sets the drawing file mode.

setPasteSpecialDrawingAutomaticExtendExposure

Description

DRAWING: This function sets the extend exposure mode and the drawing as keyframe mode

Syntax

```
copyPaste.setPasteSpecialDrawingAutomaticExtendExposure( extendExposure,
keyFrameMode );
```

Arguments

- **extendExposure**: enables the extend exposure mode.
- **keyframesMode**: enables the drawing as keyframe mode.

setPasteSpecialColorPaletteOption

Description

PALETTE: This function sets the colour palette option. See the paste special dialog.

Syntax

```
copyPaste.setPasteSpecialColorPaletteOption( "mode" );
```

Arguments

- **mode**: This is the colour palette option

Acceptable Strings are:

- ⇒ 126
- ⇒ DO_NOTHING
- ⇒ REUSE_PALETTES
- ⇒ COPY_AND_OVERWRITE_EXISTING_PALETTES
- ⇒ COPY_AND_CREATE_NEW_PALETTES
- ⇒ COPY_AND_CREATE_NEW_PALETTES_IN_ELEMENT_FOLDER
- ⇒ COPY_PALETTE_AND_MERGE_COLOURS
- ⇒ COPY_PALETTE_AND_UPDATE_COLOURS
- ⇒ LINK_TO_ORIGINAL
- ⇒ COPY_SCENE_PALETTE_AND_MERGE_COLOURS
- ⇒ COPY_SCENE_PALETTE_AND_UPDATE_COLOURS */

Element

With the Element functions, you can retrieve information about the elements in your scene and you can add and remove them.

The following is a list of the Element functions:

- `add`, on page 60
- `fieldChart`, on page 60
- `getNameById`, on page 61
- `id`, on page 61
- `numberOf`, on page 61
- `pixmapFormat`, on page 61
- `remove`, on page 62
- `renameById`, on page 62
- `scanType`, on page 62
- `vectorType`, on page 62

Example

This script adds an element named "newDrawing" to your scene.

```
function addElement
{
    var id = element.add("newDrawing", "BW", 12, "TVG", "SCAN");
}
```

add

Description

This function adds an element to the scene, and returns the element id of the newly added element if successful, otherwise it returns -1

Syntax

```
element.add("name", scanType, fieldChart, fileFormat, Vectorize);
```

Arguments

"name", scanType, fieldChart, fileFormat, Vectorize

- **name**: Any string for the name.
- **scanType**: "Colour", "Gray_Scale", "BW" (for black and white) or "None".
- **fieldChart**: 12, 16 or 24.
- **fileFormat**: "SCAN", "OPT", "PAL", "SGI", "TGA", "YUV", "OMF" or "PSD".
- **Vectorize**: "TVG" or "None".

fieldChart

Description

This function returns the field chart size for a given element. The element ID must be provided. The field chart is a number such as 12,16 or 24 field.

Syntax

```
element.fieldChart( id );
```

Arguments

- **id**: This is the ID for the desired element. See `id()` to obtain the ID for a given element.

Folder

Description

This function returns the name of the folder on disk of the element.

Syntax

```
element.folder( id );
```

Arguments

- **id**: This is the ID for the desired element. See `id()` to obtain the id for a given element.

getNameById

Description

This function returns the name of the element.

Syntax

```
element.getNameById(id);
```

Arguments

id

- **id**: This is the id of the desired element.

id

Description

This function returns the id (key) of the element.

Syntax

```
element.id(index);
```

Arguments

index

- **index**: An integer from 0 to the value returned by the function `element.numberOf`, representing the index number of the element. The id and the index are not always the same value.

numberOf

Description

This function returns the number of elements in the scene.

Syntax

```
element.numberOf();
```

Arguments

None.

pixmapFormat

Description

This function returns the pixmap format for the provided element ID.

Syntax

```
element.pixmapFormat( id );
```

Arguments

- **id**: This is the ID for the desired element. See `id()` to obtain the ID for a given element.

remove

Description

This function removes the given element. Optionally, delete the element from the disk. This function returns true when successful.

Syntax

```
element.remove( id, deleteDiskFile );
```

Arguments

- **id**: This the ID for the desired element.
- **deleteDiskFile**: A boolean that indicates remove all files. If this is false, the element is removed from the project and the database in Animate, but the files remains on the folder.

renameById

Description

This function can be used to rename the given element. Returns true when successful.

Syntax

```
element.renameById( id, newName );
```

Arguments

- **id**: This is the id for the desired element.
- **newName**: The new name of the element. This name must be unique. This operation will physically rename all the files associated to this element.

scanType

Description

This function returns a string that is the scan type of the element. The scan type is either: COLOR, GRAY_SCALE or BW

Syntax

```
element.scanType( id );
```

Arguments

- **id**: This is the ID for the desired element. See `id()` to obtain the ID for a given element.

vectorType

Description

This function returns the vector type for the given element. In theory, there is support for multiple types of vector drawing. In practice, only TVG has been implemented. The value 0 : indicates that the element is NOT a vector drawing. It is an IMAGE type.

The value 2: indicates that the element is a TVG vector drawing.

Syntax

```
element.vectorType();
```

Arguments

None

Exporter

This set of functions provides access to the project's export directory. The following is a list of the Exporter functions:

- `cleanExportDir`, on page 63
- `getExportDir`, on page 63

Example

```
var exportDir = Application.exporter.getExportDir();
var exportFile = exportDir + "paletteList.txt";

var logFile = new File(exportFile);
logFile.open(File.WriteOnly);
logFile.writeLine(scene.currentScene() + " palettes:");
logFile.writeLine("");

var numPalettes = PaletteManager.getNumPalettes();
for (i = 0; i < numPalettes; ++i)
{
    var paletteName = PaletteManager.getPaletteName(i);
    logFile.writeLine(paletteName);
}

MessageBox.information("Palette list exported to: " + exportFile);
```

cleanExportDir

Description

This function removes all files from the project export directory.

Syntax

```
exporter.cleanExportDir();
```

Arguments

None.

getExportDir

Description

This function returns the path of the project export directory.

Syntax

```
exporter.getExportDir();
```

Arguments

None.

Frame

With the Frame functions, you can retrieve values from frames and you can add and remove frames in your scene.

The following is a list of the Frame functions:

- `current`, on page 64
- `insert`, on page 65
- `numberOf`, on page 65
- `remove`, on page 65
- `setCurrent`, on page 65

Example

This example is called as soon as a scene is opened.

```
function sceneOpened()
{
// this part of the function launches the newScene function if
// the scene has only one frame
    if (frame.numberOf() == 1)
        newScene();
}

function newScene()
{
// this function opens a dialog box named Create New Scene, which
// allows users to enter the number of frames to add to the scene
    var d = new Dialog;
    d.title = "Create New Scene";
    var nbFrames = new SpinBox;
    nbFrames.label = "Number of Frames";
    nbFrames.minimum = 1;
    nbFrames.maximum = 300;
    nbFrames.value = 60;
    d.add(nbFrames);

    if (d.exec())
    {
        var oldNbFrames = frame.numberOf();
        frame.insert(0, nbFrames.value - oldNbFrames);
    }
}
```

current

Description

This function returns the number of the current frame.

Syntax

```
frame.current();
```

Arguments

None.

insert

Description

This function inserts frames at the selected frame number.

Syntax

```
frame.insert(atFrame, nbFrames);
```

Arguments

atFrame, **nbFrames**

- **atFrame**: Integer for the frame number at which the frames will be inserted. Frames are inserted after the frame indicated. Use 0 to insert frames before the first frame.
- **nbFrames**: Integer for the number of frames to insert.

numberOf

Description

This function returns the number of frames in the scene.

Syntax

```
frame.numberOf();
```

Arguments

None.

remove

Description

This function deletes frames starting from the selected frame number.

Syntax

```
frame.remove(startFrame, nbFrames);
```

Arguments

startFrame, **nbFrames**

- **startFrame**: Integer indicating at which frame to start removing frames. Use 0 to delete frames from the beginning.
- **nbFrames**: Integer for the number of frames to remove.

setCurrent

Description

This function allows you to change the current frame.

Syntax

```
frame.setCurrent( frame );
```

Arguments

- **Frame**: The new current frame.

Function Curve

With the Function Curve functions, you can retrieve and modify values in the function curves.

The following is a list of the Function Curve functions:

- `addCtrlPointAfter3DPath`, on page 68
- `addKeyFrame3DPath`, on page 68
- `angleEaseIn`, on page 68
- `angleEaseOut`, on page 69
- `holdStartFrame`, on page 69
- `holdStep`, on page 69
- `holdStopFrame`, on page 69
- `numberOfPoints`, on page 70
- `numberOfPoints3DPath`, on page 70
- `pointBias3DPath`, on page 70
- `pointConstSeg`, on page 70
- `pointContinuity`, on page 71
- `pointContinuity3DPath`, on page 71
- `pointEaseIn`, on page 71
- `pointEaseOut`, on page 71
- `pointHandleLeftX`, on page 72
- `pointHandleLeftY`, on page 72
- `pointHandleRightX`, on page 72
- `pointHandleRightY`, on page 72
- `pointLockedAtFrame`, on page 73
- `pointTension3DPath`, on page 73
- `pointX`, on page 73
- `pointX3DPath`, on page 73
- `pointY`, on page 74
- `pointY3DPath`, on page 74
- `pointZ3DPath`, on page 74
- `removePoint3DPath`, on page 74
- `setBezierPoint`, on page 75
- `setEasePoint`, on page 75
- `setHoldStartFrame`, on page 76
- `setHoldStopFrame`, on page 76
- `setHoldStep`, on page 76
- `setPoint3DPath`, on page 76
- `setVeloBasedPoint`, on page 77

Example

The following script adds three function columns and sets the values for various points on the function curves.

```
function addSetFunctions()
{
    /* creates the function columns if needed*/
    if ( !column.add("3DPATH_FUNC", "3DPATH") )
        System.println( "Error creating 3dpath column" );

    if ( !column.add("BEZIER_FUNC", "BEZIER") )
        System.println( "Error creating bezier column" );
}
```

```

if ( !column.add("EASE_FUNC", "EASE" )
System.println( "Error creating ease column" );

// adds 2 keyframes on a 3DPATH at frame 6 and 12
// with values x=2, y=2, z=1, tension=3, continuity=-1, bias=1
func.addKeyFrame3DPath( "3DPATH_FUNC", 6, 2, 2, 1, 3, -1, 1 );
func.addKeyFrame3DPath( "3DPATH_FUNC", 12, 2, 2, 1, 3, -1, 1
);

// removes a keyframe on 3DPATH at frame 12
func.removePoint3DPath( "3DPATH_FUNC", 12 );

// prints the number of points on a 3DPath column
var pathPoints = func.numberOfPoints3DPath("3DPATH_FUNC");
System.println( "3DPATH_FUNC column has " + pathPoints + "
points" );

// adds a keyframe on a BEZIER at frame 10 with values y=5,
// handle_leftx=9, handle_lefty=5, handle_rightx=12,
// handle_righty=-0.5, constant seg=false, continuity=smooth
if ( ! func.setBezierPoint( "BEZIER_FUNC", 10, 5, 9, 5, 12, -
0.5, false, "SMOOTH" ) )
System.println( "Error creating point on bezier" );

// sets the step value for a function column starting at frame
// 1 upto frame 15 hold 2
func.setHoldStartFrame("BEZIER_FUNC", 1);
func.setHoldStopFrame("BEZIER_FUNC", 15);
func.setHoldStep("BEZIER_FUNC", 2);

// prints the number of points on a function column
var bezPoints = func.numberOfPoints("BEZIER_FUNC");
System.println( "BEZIER_FUNC column has " + bezPoints + "
points" );

// adds a keyframe on an EASE at frame 10 with values y=5,
// ease_inx=0, ease_iny=0, ease_outx=0, ease_outy=180
// constant seg=true, continuity=straight
if ( ! func.setEasePoint( "EASE_FUNC", 10, 6, 0, 0, 0, 180,
true, "STRAIGHT" ) )
System.println( "Error creating point on ease" );

}

```

addCtrlPointAfter3DPath

Description

This function adds a keyframe after a point on a 3D Path and sets the X, Y and Z values, as well as the tension, continuity and bias.

Syntax

```
func.addCtrlPointAfter3DPath("columnName", point, X, Y, Z, tension, continuity, bias);
```

Arguments

"columnName", point, X, Y, Z, tension, continuity, bias

- **columnName**: The name of the column.
- **point**: The number of the point on the curve, from 0 to **n**, where **n** is the total number of points. The last point on the curve is **n-1**.
- **X**: X value for the point on the 3D Path.
- **Y**: Y value for the point.
- **Z**: Z value for the point.
- **tension**: The tension value of the keyframe.
- **continuity**: The continuity value of the keyframe.
- **bias**: The bias value of the keyframe.

addKeyFrame3DPath

Description

This function adds a keyframe to a 3D Path and sets the X, Y and Z value, as well as the tension, continuity and bias.

Syntax

```
func.addKeyFrame3DPath("columnName", frame, X, Y, Z, tension, continuity, bias);
```

Arguments

"columnName", frame, X, Y, Z, tension, continuity, bias

- **columnName**: The name of the column.
- **frame**: Frame number for the point.
- **X**: X value for the point on the 3D Path.
- **Y**: Y value for the point.
- **Z**: Z value for the point.
- **tension**: The tension value of the keyframe.
- **continuity**: The continuity value of the keyframe.
- **bias**: The bias value of the keyframe.

angleEaseIn

Description

This function returns the angle of the ease-in handle.

Syntax

```
func.angleEaseIn("columnName", point);
```

Arguments

"columnName", point

- **columnName**: The name of the column.
- **point**: The number of the point on the curve, from 0 to **n**, where **n** is the total number of points. The last point on the curve is **n-1**.

angleEaseOut

Description

This function returns the angle of the ease-out handle.

Syntax

```
func.angleEaseOut("columnName", point);
```

Arguments

"columnName", point

- **columnName**: The name of the column.
- **point**: The number of the point on the curve, from 0 to **n**, where **n** is the total number of points. The last point on the curve is **n-1**.

holdStartFrame

Description

This function returns the Start value from the Hold Value Editor dialog box, for Bezier, Ease and Velo-based Function Editors.

Syntax

```
func.holdStartFrame("columnName");
```

Arguments

"columnName"

- **columnName**: The name of the column.

holdStep

Description

This function returns the Step value from the Hold Value Editor dialog box, for Bezier, Ease and Velo-based Function Editors.

Syntax

```
func.holdStep("columnName");
```

Arguments

"columnName"

- **columnName**: The name of the column.

holdStopFrame

Description

This function returns the Stop value from the Hold Value Editor dialog box, for Bezier, Ease and Velo-based Function Editors.

Syntax

```
func.holdStopFrame("columnName");
```

Arguments

"columnName"

- **columnName**: The name of the column.

numberOfPoints

Description

This function returns the number of keyframes and control points on a curve.

Syntax

```
func.numberOfPoints("columnName");
```

Arguments

"columnName"

- **columnName**: The name of the column.

numberOfPoints3DPath

Description

This function returns the number of keyframes and control points on the 3D Path.

Syntax

```
func.numberOfPoints3DPath("columnName");
```

Arguments

"columnName"

- **columnName**: The name of the column.

pointBias3DPath

Description

This function returns the bias value for the specified point on the 3D Path.

Syntax

```
func.pointBias3DPath("columnName", point);
```

Arguments

"columnName", point

- **columnName**: The name of the column.
- **point**: The number of the point on the curve, from 0 to **n**, where **n** is the total number of points. The last point on the curve is **n-1**.

pointConstSeg

Description

This function returns a 1 (one) to indicate that the point is on a constant segment, or a 0 (zero) to indicate that the point is not on a constant segment.

Syntax

```
func.pointConstSeg("columnName", point);
```

Arguments

"columnName", point

- **columnName**: The name of the column.
- **point**: The number of the point on the curve, from 0 to **n**, where **n** is the total number of points. The last point on the curve is **n-1**.

pointContinuity

Description

This function returns the continuity of the curve that follows the point. One of the following values will be returned, in upper-case: SMOOTH, CORNER or STRAIGHT.

Syntax

```
func.pointContinuity("columnName", point);
```

Arguments

"columnName", point

- **columnName**: The name of the column.
- **point**: The number of the point on the curve, from 0 to **n**, where **n** is the total number of points. The last point on the curve is **n-1**.

pointContinuity3DPath

Description

This function returns the continuity value (STRAIGHT, SMOOTH or CORNER) for the specified point on the 3D Path.

Syntax

```
func.pointContinuity3DPath("columnName", point);
```

Arguments

"columnName", point

- **columnName**: The name of the column.
- **point**: The number of the point on the curve, from 0 to **n**, where **n** is the total number of points. The last point on the curve is **n-1**.

pointEaseIn

Description

This function returns the number of frames in the ease-in.

Syntax

```
func.pointEaseIn("columnName", point);
```

Arguments

"columnName", point

- **columnName**: The name of the column.
- **point**: The number of the point on the curve, from 0 to **n**, where **n** is the total number of points. The last point on the curve is **n-1**.

pointEaseOut

Description

This function returns the number of frames in the ease-out.

Syntax

```
func.pointEaseOut("columnName", point);
```

Arguments

"columnName", point

- **columnName**: The name of the column.
- **point**: The number of the point on the curve, from 0 to **n**, where **n** is the total number of points. The last point on the curve is **n-1**.

pointHandleLeftX

Description

This function returns the X value of the left handle of a point on a curve.

Syntax

```
func.pointHandleLeftX("columnName", point);
```

Arguments

"columnName", point

- **columnName**: The name of the column.
- **point**: The number of the point on the curve, from 0 to **n**, where **n** is the total number of points. The last point on the curve is **n-1**.

pointHandleLeftY

Description

This function returns the Y value of the left handle of a point on a curve.

Syntax

```
func.pointHandleLeftY("columnName", point);
```

Arguments

"columnName", point

- **columnName**: The name of the column.
- **point**: The number of the point on the curve, from 0 to **n**, where **n** is the total number of points. The last point on the curve is **n-1**.

pointHandleRightX

Description

This function returns the X value of the right handle of a point on a curve.

Syntax

```
func.pointHandleRightX("columnName", point);
```

Arguments

"columnName", point

- **columnName**: The name of the column.
- **point**: The number of the point on the curve, from 0 to **n**, where **n** is the total number of points. The last point on the curve is **n-1**.

pointHandleRightY

Description

This function returns the Y value of the right handle of a point on a curve.

Syntax

```
func.pointHandleRightY("columnName", point);
```

Arguments

"columnName", point

- **columnName**: The name of the column.
- **point**: The number of the point on the curve, from 0 to **n**, where **n** is the total number of points. The last point on the curve is **n-1**.

pointLockedAtFrame

Description

This function returns the frame the point is locked at or 0 if the point is not locked.

Syntax

```
func.pointLockedAtFrame("columnName", point);
```

Arguments

"columnName", point

- **columnName**: The name of the column.
- **point**: The number of the point on the curve, from 0 to **n**, where **n** is the total number of points. The last point on the curve is **n-1**.

pointTension3DPath

Description

This function returns the tension value for the specified point on the 3D Path.

Syntax

```
func.pointTension3DPath("columnName", point);
```

Arguments

"columnName", point

- **columnName**: The name of the column.
- **point**: The number of the point on the curve, from 0 to **n**, where **n** is the total number of points. The last point on the curve is **n-1**.

pointX

Description

This function returns the X value (frame number) of a point on a function curve.

Syntax

```
func.pointX("columnName", point);
```

Arguments

"columnName", point

- **columnName**: The name of the column.
- **point**: The number of the point on the curve, from 0 to **n**, where **n** is the total number of points. The last point on the curve is **n-1**.

pointX3DPath

Description

This function returns the value of the specified point on the X path.

Syntax

```
func.pointX3DPath("columnName", point);
```

Arguments

"columnName", point

- **columnName**: The name of the column.
- **point**: The number of the point on the curve, from 0 to **n**, where **n** is the total number of points. The last point on the curve is **n-1**.

pointY

Description

This function returns the Y value of a point on a function curve.

Syntax

```
func.pointY("columnName", point);
```

Arguments

"columnName", point

- **columnName**: The name of the column.
- **point**: The number of the point on the curve, from 0 to **n**, where **n** is the total number of points. The last point on the curve is **n-1**.

pointY3DPath

Description

This function returns the value of the specified point on the Y path.

Syntax

```
func.pointY3DPath("columnName", point);
```

Arguments

"columnName", point

- **columnName**: The name of the column.
- **point**: The number of the point on the curve, from 0 to **n**, where **n** is the total number of points. The last point on the curve is **n-1**.

pointZ3DPath

Description

This function returns the value of the specified point on the Z path.

Syntax

```
func.pointZ3DPath("columnName", point);
```

Arguments

"columnName", point

- **columnName**: The name of the column.
- **point**: The number of the point on the curve, from 0 to **n**, where **n** is the total number of points. The last point on the curve is **n-1**.

removePoint3DPath

Description

This function removes either a control point or a keyframe from the 3D Path.

Syntax

```
func.removePoint3DPath("columnName", point);
```

Arguments

"columnName", point

- **columnName**: The name of the column.
- **point**: The number of the point on the curve, from 0 to **n**, where **n** is the total number of points. The last point on the curve is **n-1**.

setBezierPoint

Description

This function sets the values of a point on a Bezier function curve.

Syntax

```
func.setBezierPoint("columnName", frame, Y, handleLeftX, handleLeftY,
  handleRightX, handleRightY, constSeg, "continuity");
```

Arguments

"columnName", frame, Y, handleLeftX, handleLeftY, handleRightX, handleRightY, constSeg, "continuity"

- **columnName**: The name of the column.
- **frame**: Frame number for the point.
- **Y**: Y value for the point.
- **handleLeftX**: X value for the left handle of the point.
- **handleLeftY**: Y value for the left handle.
- **handleRightX**: X value for the right handle.
- **handleRightY**: Y value for the right handle.
- **constSeg**: Boolean expression (with a true or false value) to indicate whether the segment is constant or interpolated.
- **"continuity"**: String value for the continuity of the point. The string must be in all upper-case. The following are the acceptable values: STRAIGHT, SMOOTH and CORNER.

setEasePoint

Description

This function sets the values of a point on an Ease function curve.

Syntax

```
func.setEasePoint("columnName", frame, Y, easeIn, angleEaseIn, easeOut,
  angleEaseOut, constSeg, "continuity");
```

Arguments

"columnName", frame, Y, easeIn, angleEaseIn, easeOut, angleEaseOut, constSeg, "continuity"

- **columnName**: The name of the column.
- **frame**: Frame number for the point.
- **Y**: Y value for the point.
- **easeIn**: The number of frames in the ease-in.
- **angleEaseIn**: The angle of the ease-in handle.
- **easeOut**: The number of frames in the ease-out.
- **angleEaseOut**: The angle of the ease-out handle.
- **constSeg**: Boolean expression (with a true or false value) to indicate whether the segment is constant or interpolated.
- **"continuity"**: String value for the continuity of the point. The string must be in all upper-case. The following are the acceptable values: STRAIGHT, SMOOTH and CORNER.

setHoldStartFrame

Description

This function sets the Start value in the Hold Value Editor dialog box, for Bezier, Ease and Velo-based Function Editors.

Syntax

```
func.setHoldStartFrame("columnName", startFrame);
```

Arguments

"columnName", startFrame

- **columnName**: The name of the column.
- **startFrame**: Integer for the start frame of the hold.

setHoldStep

Description

This function sets the Hold value in the Hold Value Editor dialog box, for Bezier, Ease and Velo-based Function Editors.

Syntax

```
func.setHoldStep("columnName", step);
```

Arguments

"columnName", stepNumber

- **columnName**: The name of the column.
- **stepNumber**: Integer for the value of the steps in the hold.

setHoldStopFrame

Description

This function sets the Stop value in the Hold Value Editor dialog box, for Bezier, Ease and Velo-based Function Editors.

Syntax

```
func.setHoldStopFrame("columnName", stopFrame);
```

Arguments

"columnName", stop

- **columnName**: The name of the column.
- **stopFrame**: Integer for the stop frame of the hold.

setPoint3DPath

Description

This function sets the properties of a point on a 3D Path, including X, Y, and Z values, and tension, continuity and bias.

Syntax

```
func.setPoint3DPath("columnName", point, X, Y, Z, tension, continuity, bias);
```

Arguments

"columnName", point, X, Y, Z, tension, continuity, bias

- **columnName**: The name of the column.
- **point**: The number of the point on the curve, from 0 to n, where n is the total number of points. The last point on the curve is n-1.
- **X**: X value for the point on the 3D Path.
- **Y**: Y value for the point.
- **Z**: Z value for the point.
- **tension**: The tension value of the keyframe.
- **continuity**: The continuity value of the keyframe.
- **bias**: The bias value of the keyframe.

setVeloBasedPoint

Description

This function sets the values of a point on a Velocity-Based function curve.

Syntax

```
func.setVeloBasedPoint("columnName", frame, Y);
```

Arguments

`"columnName"`, `frame`, `Y`

- `columnName`: The name of the column.
- `frame`: Frame number for the point.
- `Y`: Y value for the point.

MessageLog

This set of functions allows the user to print messages in the message log window.

The following is a list of the mssagelog functions:

- `debug`, on page 78
- `isDebug`, on page 78
- `setDebug`, on page 78
- `trace`, on page 78

Example

```
MessageLog.trace("Export template failed. Nothing selected.");
```

debug

Description

This function writes the message to the message log if debug mode is on.

Syntax

```
MessageLog.debug( message );
```

Arguments

- `message`: is the message string.

isDebug

Description

This function returns whether debug mode is enabled in the messageLog.

Syntax

```
MessageLog.isDebug();
```

Arguments

None

setDebug

Description

This function sets the debug mode in message log. This will set debug mode for the entire application.

Syntax

```
MessageLog.setDebug( b );
```

Arguments

- `b`: is a boolean (true or false)

trace

Description

This function writes the message to the message log.

Syntax

```
MessageLog.trace( message );
```

Arguments

`message`: is a string.

Node

Nodes are synonymous with Filter, IO, Move and Group Modules in the Network View.

With Node functions, you can retrieve values from modules in your effects and compositing network. You can also use these functions to add and link modules in your network, and set their attributes.

Because there can be identical module names in the effects network, you must use the full path to identify a module. Similarly, when functions return module names, they often use the full path.

The syntax for module paths is:

Top/Group_Name/Module_Name

NOTE:

To find the proper syntax for modules and their attributes, create a scene, add the modules you want to check, save the scene and then check the *.stage file.

The following is a list of the Node functions:

- add, on page 81
- addCompositeToGroup, on page 81
- coordX, on page 81
- coordY, on page 81
- createGroup, on page 82
- deleteNode, on page 82
- dstNode, on page 82
- equals, on page 82
- explodeGroup, on page 83
- flatDstNode, on page 83
- flatSrcNode, on page 83
- getCameras, on page 83
- getDefaultCamera, on page 84
- getEnable, on page 84
- getMatrix, on page 84
- getName, on page 84
- getTextAttr, on page 84
- isGroup, on page 85
- isLinked, on page 85
- link, on page 85
- linkAttr, on page 85
- linkedColumn, on page 86
- noNode, on page 86
- numberOfInputPorts, on page 86
- numberOfOutputLinks, on page 86
- numberOfOutputPorts, on page 87
- numberOfSubNodes, on page 87
- parentNode, on page 87
- rename, on page 87
- root, on page 88
- setAsDefaultCamera, on page 88
- setAsGlobalDisplay, on page 88
- setEnable, on page 88
- setCoord, on page 88
- setGlobalToDisplayAll, on page 89
- setTextAttr, on page 89
- srcNode, on page 89
- subNode, on page 89

- `subNodeByName`, on page 90
- `type`, on page 90
- `unlink`, on page 90
- `unlinkAttr`, on page 90
- `width/height`, on page 91

Example

This script automates the process of compositing several modules together using a Composite Module. This script takes all of the selected modules in the Network View and links them to a new Composite Module, named "comp".

```
{
/* The parentNode function in this script adds the Composite Module in the parent
group of the selected modules.
The link function, in a loop, connects all selected modules to the new Composite
Module.
The setCoord function positions the new module in the network.
function compose()*/

    var n = selection.numberOfNodesSelected();
    var i, posx, posy;

    if (n > 0)
    {
        var comp = node.add(
            node.parentNode(selection.selectedNode(0)), "comp",
            "COMPOSITE", 0, 0, 0);
        posx = 0; posy = -10000;
        for (i = 0; i < n; ++i)
        {
            var selNode = selection.selectedNode(i);
            node.link(selNode, 0, comp, i);
            posx += node.coordX(selNode);
            if (node.coordY(selNode) > posy)
                posy = node.coordY(selNode);
        }
        posx /= n;
        posy += 50;
        node.setCoord(comp, posx, posy)
    }
}
```

add

Description

This function adds a module to the network.

Syntax

```
node.add("parent_group_path", "name", "type", X, Y, Z);
```

Arguments

"parent_group_path", "name", "type", X, Y, Z

- **parent_group_path**: The path of the parent node into which you want to add this module.
- **name**: The name of the module you will add.
- **type**: The type of module you will add. The module types are like: READ, COMPOSITE, PEG, QUADMAP, SHADOW, etc.
- **x**: The X position of the module in the Network View.
- **y**: The Y position of the module in the Network View.
- **z**: The Z position of the module in the Network View. This property is important when two modules overlap.

addCompositeToGroup

Description

This function returns the value of the "Add Composite To Group" attribute of the peg-module. This method is obsolete.

Syntax

```
node.addCompositeToGroup();
```

Arguments

None

coordX

Description

This function returns an integer indicating the X position of a module in the network.

Syntax

```
node.coordX("node_path");
```

Arguments

"node_path"

- **node_path**: The path of the node whose X position you want to identify.

coordY

Description

This function returns an integer indicating the Y position of a module in the network.

Syntax

```
node.coordY("node_path");
```

Arguments

"node_path"

- **node_path**: The path of the node whose Y position you want to identify.

createGroup

Description

Create a group from the selection of nodes. The list of nodes is a string where the nodes are separated by commas. The actual name of each node in the list of nodes must include the full path of that node. The function returns the full path of the created group, or an empty string if the creation of the node failed.

The parent of the group is implicitly specified. It will be the same as the parent of the first node in the list.

Syntax

```
node.createGroup( nodes, groupName );
```

Arguments

- **nodes**: This a list comma delimited list of names of nodes.
- **groupName**: The name of the group to create. The actual group.

deleteNode

Description

Delete a single node. Optionally, delete all columns and element associated to that node. The column and element would only be removed when no other modules refer to them. This function returns true when successful.

Syntax

```
node.deleteNode( name, isDeleteColumn, isDeleteElement );
```

Arguments

- **name**: The full path of the node to delete.
- **isDeleteColumn**: A boolean that indicates if the columns referenced by that node shall be removed if they are no longer in use.
- **isDeleteElement**: A boolean that indicates if the element should also be removed, if they are no longer in use.

dstNode

Description

This function returns the path of the destination module linked to by the output port on the source module.

Syntax

```
node.dstNode("node_path", iPort, iLink);
```

Arguments

"node_path", iPort, iLink

- **node_path**: The path of the module whose output you want to get.
- **iPort**: The port number on which you want to find the connected module. This value is between 0 and the results of the `numberOfInputPorts` function.
- **iLink**: The link number whose destination module you want to find. This value is between 0 and the results of the `numberOfInputLinks` function.

equals

Description

This function returns true or false to indicate if a node path is equal to another. Used to compare node paths.

Syntax

```
node.equals("node_path", "node_path");
```

Arguments

"node_path", "node_path"

- **node_path**: The path of the modules you want to compare.

explodeGroup

Description

Explode a group into its parent group. This function is identical to the "Explode Selected Group" from the Network view.

This function returns true if successful.

Syntax

```
node.explodeGroup( nameOfGroup );
```

Arguments

- **nameOfGroup**: The full path name of a group

flatDstNode

Description

If the `dstNode` is a Group Module, this function returns the path of the module inside the Group Module that is the destination. This function behaves like the `dstNode` function when the module is not in a group.

Syntax

```
node.flatDstNode("node_path", iPort, iLink);
```

Arguments

"node_path", iPort, iLink

- **node_path**: The path of the module whose output you want to get.
- **iPort**: the port number on which you want to find the connected module. This value is between 0 and the results of the `numberOfInputPorts` function.
- **iLink**: The number of the link whose destination module you want to find. This value is between 0 and the results of the `numberOfInputLinks` function.

flatSrcNode

Description

If the `srcNode` is a Group Module, this function returns the path of the module inside the Group Module that is the source. If the source is not inside a group, the `flatSrcNode` function behaves like the `srcNode` function.

Syntax

```
node.flatSrcNode("node_path", iPort);
```

Arguments

"node_path", iPort

- **node_path**: The path of the module whose input ports you want to get.
- **iPort**: The port number on which you want to find the module that is connected to it. This value is between 0 and the results of the `numberOfInputPorts` function.

getCameras

Description

Returns a list of all cameras within the scene.

Syntax

```
node.getCameras();
```

Arguments

- None

getDefaultCamera

Description

This function returns the name of the default camera.

Syntax

```
node.getDefaultCameras();
```

Arguments

None

getEnable

Description

This function returns whether a module is enabled or not.

Syntax

```
node.getEnable(nodeName);
```

Arguments

- **nodeName**: is the string name for the node.

getMatrix

Description

This function returns the model matrix of a node.

Syntax

```
node.getmatrix(nodeName, frame);
```

Arguments

- **nodeName**: is the string name for the node.
- **Frame**: is the int that specifies the frame number

getName

Description

This function returns the name of a module.

Syntax

```
node.getName("node_path");
```

Arguments

"node_path"

- **node_path**: The path of the node that you want to get the name of.

getTextAttr

Description

This function returns the value(s) of the module's selected attribute(s).

Syntax

```
node.getTextAttr("node_path", atFrame, "attrName");
```

Arguments

"node_path", atFrame, "attrName"

- **node_path**: The path of the module whose attributes you want.
- **atFrame**: The frame number from which you want to extract the attribute value. If the value is static, you must still pass a value. You can, for example, pass 1 to take the value at the first frame.
- **attrName**: The attribute whose value you want.

isGroup

Description

This function returns a true or false value indicating if the module is a Group Module.

Syntax

```
node.isGroup("node_path");
```

Arguments

"node_path"

- **node_path**: The path of the node that you want to test to see if it is a Group Module.

isLinked

Description

This function returns true or false to indicate if a port is connected to another module.

Syntax

```
node.isLinked("node_path", iPort);
```

Arguments

"node_path", iPort

- **node_path**: The path of the module whose ports you want to check.
- **iPort**: The number of the port whose link status you want to check. This value is between 0 and the results of the `numberOfInputPorts` function.

link

Description

This function links a port on a module to a port on another module.

Syntax

```
node.link("srcNode_path", srcPort, "dstNode_path", dstPort);
```

Arguments

"srcNode_path", srcPort, "dstNode_path", dstPort

- **srcNode_path**: The path of the module whose output port you want to link to a destination module.
- **srcPort**: The port that you want to link to the input port on the destination module. This value is between 0 and the results of the `numberOfOutputPorts` function.
- **dstNode_path**: The path of the module whose input port you want to link to the source module.
- **dstPort**: The port on the destination module that you want to link to the output port from the source module. This value is between 0 and the results of the `numberOfInputPorts` function.

linkAttr

Description

This function links an attribute to a function column in the Xsheet View.

Syntax

```
node.linkAttr("node_path", "attrName", "columnName");
```

Arguments

"node_path", "attrName", "columnName"

- **node_path**: The path of the module whose attribute you want to link to a function column.
- **attrName**: The name of the attribute that you want to link.
- **columnName**: The name of the column that you want to link to the attribute.

linkedColumn

Description

This function returns the name of the column that an attribute is linked to. If the attribute is not linked to a column, the function returns the null string.

Syntax

```
node.linkedColumn("node_path", "attrName");
```

Arguments

"node_path", "attrName"

- **node_path**: The path of the module you want to check.
- **attrName**: The attribute you want to check to see if it is linked to a column.

noNode

Description

This function returns the null string that is returned by other functions when there is an error.

Syntax

```
node.noNode();
```

Arguments

None.

numberOfInputPorts

Description

This function returns an integer indicating the number of input ports on the module.

Syntax

```
node.numberOfInputPorts("node_path");
```

Arguments

"node_path"

- **"node_path"**: The path of the module whose input ports you want to count.

numberOfOutputLinks

Description

This function returns an integer indicating the number of modules actually linked from the output ports.

Syntax

```
node.numberOfOutputLinks("node_path", iPort);
```

Arguments

"node_path", iPort

- **node_path**: The path of the module whose output ports you want to check.
- **iPort**: The port number on which you want to locate the module that is connected to it. This value is between 0 and the results of the **numberOfInputPorts** function.

numberOfOutputPorts

Description

This function returns an integer indicating the number of output ports on a module.

Syntax

```
node.numberOfOutputPorts ("node_path");
```

Arguments

"node_path"

- **node_path**: The path of the module whose output ports you want to check.

numberOfSubNodes

Description

This function returns an integer that indicates the number of modules contained in a group.

Syntax

```
node.numberOfSubNodes ("node_path");
```

Arguments

"node_path"

- **node_path**: The path of the Group Module that you want to query to count the modules it contains.

parentNode

Description

This function returns the path of the parent level of a module contained in a group.

Syntax

```
node.parentNode ("node_path");
```

Arguments

"node_path"

- **"node_path"**: The path of the module whose parent you want to locate.

rename

Description

This function changes the name of a module.

Syntax

```
node.rename ("node_path", "newName");
```

Arguments

"node_path", "newName"

- **node_path**: The path of the module whose name you want to change.
- **newName**: The new name for the module.
- **atFrame**: The frame number at which you want to change the value. If the value is static, you must still pass a value.
- **attrValue**: The new value for the attribute.

root

Description

This function returns the name of the Top level in the network, which is "Top".

Syntax

```
node.root();
```

Arguments

None.

setAsDefaultCamera

Description

This function sets the default camera in the scene. Returns whether successful or not.

Syntax

```
node.setAsDefaultCamera(cameraName);
```

Arguments

- **cameraName**: is the string name of the camera.

setAsGlobalDisplay

Description

This function changes the global display used in the application. The node must be the full path of a display module.

This function returns true if successful. It returns false if the display node was not found or an invalid name was provided.

Syntax

```
node.setAsGlobalDisplay( displayNodeName );
```

Arguments

- **displayNodeName**: The full path of a display node

setCoord

Description

This function sets the position of a module in the network.

Syntax

```
node.setCoord("node_path", X, Y);
```

Arguments

"node_path", X, Y

- **node_path**: The path of the module you want to reposition in the Network View.
- **X**: The X position of the module in the Network View.
- **Y**: The Y position of the module in the Network View.

setEnabled

Description

This function sets the enable flag of the node.

Syntax

```
node.setEnabled(nodeName);
```

Arguments

- **nodeName**: is the string name of the node.

setGlobalToDisplayAll

Description

This function changes the global display used by the application to "Display All" pseudo-display. This function returns true if successful.

Syntax

```
node.setGlobalToDisplayAll();
```

Arguments

None

setTextAttr

Description

This function changes the value of an attribute in a module.

Syntax

```
node.setTextAttr("node_path", "attrName", atFrame, "attrValue");
```

Arguments

"node_path", "attrName", atFrame, "attrValue"

- **node_path**: The path of the module whose attribute you want to change.
- **attrName**: The name of the attribute whose value you want to change.

srcNode

Description

This function returns the path for the module that the port is linked to.

Syntax

```
node.srcNode("node_path", iPort);
```

Arguments

"node_path", iPort

- **node_path**: The path of the module whose input ports you want to get.
- **iPort**: The port number on the destination module whose source module you want to find. This value is between 0 and the results of the `numberOfInputPorts` function.

subNode

Description

This function returns the path of a module in a group. Modules are counted starting with zero.

Syntax

```
node.subNode("node_path_parent", iSubNode);
```

Arguments

"node_path_parent", iSubNode

- **node_path_parent**: The path of the parent group that contains the module you want to identify.
- **iSubNode**: An integer representing the numerical value of the module. This value must be between 0 and the `numberOfSubNodes` function for that point.

subNodeByName

Description

This function returns the full path name of a child node belonging to a parent group. This function will validate that the parent group exists and that the child node exists in that parent. This function should be used instead of manually concatenating the full path name of child nodes.

Syntax

```
node.subNodeByName( parentGroup, nodeName );
```

Arguments

- **parentGroup**: Fully qualified path of the parent group.
- **nodeName**: The actual node name suffix. This one is only the short name, not the full path of the node.

type

Description

This function returns the module type. These are all of the built-in module types available from the Stage Module like: READ, COMPOSITE, PEG, QUADMAP, SHADOW, etc.

Syntax

```
node.type( "node_path" );
```

Arguments

"node_path"

- **node_path**: The path of the node whose type you want to know.

unlink

Description

This function unlinks a port on one module from the port on another module.

Syntax

```
node.unlink( "dstNode_path", inPort );
```

Arguments

"dstNode_path", inPort

- **dstNode_path**: The path of the module whose input port you want to unlink from the source module.
- **inPort**: The input port that you want to unlink. This value is between 0 and the results of the `numberOfInputPorts` function.

unlinkAttr

Description

This function unlinks an attribute from a function column.

Syntax

```
node.unlinkAttr( "node_path", "attrName" );
```

Arguments

"node_path", "attrName"

- **node_path**: The path of the module whose attribute you want to unlink from a function column.
- **attrName**: The name of the attribute that you want to unlink.

width/height

Description

This function returns the width or the height of a given node. This width / height parameter is useful for computing the position of nodes in the Network view. It uses the same unit system as the `coordX()` and `coordY()` functions.

Syntax

```
node.width( nodeName );  
node.height( nodeName )
```

Arguments

- `nodeName`: A full path name of a node.

PaletteManager

This set of functions is used to query information from the Colour View. All of these are used.

The following is a list of the PaletteManager functions:

- `getCurrentColorId`, on page 92
- `getCurrentColorName`, on page 92
- `getCurrentPalettId`, on page 92
- `getCurrentPaletteName`, on page 93
- `setCurrentPaletteByld`, on page 93
- `setCurrentColorByld`, on page 93
- `setCurrentPaletteAndColorByld`, on page 93
- `getCurrentPaletteSize`, on page 93
- `getColorName`, on page 94
- `getColorId`, on page 94
- `getNumPalettes`, on page 94
- `getNumPalettes`, on page 94
- `getPaletteName`, on page 94
- `getPaletteName`, on page 95
- `getPalettId`, on page 95
- `getPalettId`, on page 95

`getCurrentColorId`

Description

This function returns the current color Id from the ColourView.

Syntax

```
PaletteManager.getCurrentColorId();
```

Arguments

None

`getCurrentColorName`

Description

This function returns the current colour name from the ColourView.

Syntax

```
PaletteManager.getCurrentColorName();
```

Arguments

None

`getCurrentPalettId`

Description

This function returns the id of the current palette from the ColourView.

Syntax

```
PaletteManager.getCurrentPaletteId();
```

Arguments

None

getCurrentPaletteName

Description

This function returns the current palette name from the ColourView.

Syntax

```
PaletteManager.getCurrentPaletteName();
```

Arguments

None

setCurrentPaletteById

Description

This function sets the current palette in the ColourView.

Syntax

```
PaletteManager.setCurrentPaletteById( "palette" );
```

Arguments

- `palette`: This is the name of the palette

setCurrentColorById

Description

This function sets the current color in the ColourView.

Syntax

```
PaletteManager.setCurrentColorById( "color" );
```

Arguments

- `color`: This is the name of the colour

setCurrentPaletteAndColorById

Description

This function sets the current palette and colour in the ColourView.

Syntax

```
PaletteManager.setCurrentPaletteAndColorById( "palette", "color" );
```

Arguments

- `palette`: This is the palette id
- `color`: This is the colour id

getCurrentPaletteSize

Description

This function returns the size of the currently selected palette in the ColourView.

Syntax

```
PaletteManager.getCurrentPaletteSize();
```

Arguments

None

getColorName

Description

This function returns the name of the the colour in the currently selected palette.

Syntax

```
PaletteManager.getColorName(int index);
```

Arguments

- **index**: This is the colour index in the palette

getColorId

Description

This function retrieves the id of the currently selected colour.

Syntax

```
PaletteManager.getColorId(int index);
```

Arguments

- **index**: This is the colour index in the palette

getNumPalettes

Description

This function returns the number of palettes in the current selected palette list in ColourView list.

Syntax

```
PaletteManager.getNumPalettes();
```

Arguments

None

getNumPalettes

Description

This function returns the number of palettes in palette list in ColourView.

Syntax

```
PaletteManager.getNumPalettes( scenePaletteList );
```

Arguments

- **scenePaletteList**: This determines whether to check the scene palette list or the element palette list.

getPaletteName

Description

This function returns the name of the palette in the current palette list in the ColourView.

Syntax

```
PaletteManager.getPaletteName(int index);
```

Arguments

- **index**: This is the index of the palette within the palette list

getPaletteName

Description

This function returns the name of the palette in the current palette list in the ColourView.

Syntax

```
PaletteManager.getPaletteName(int index, scenePaletteList);
```

Arguments

- **index**: This is the index of the palette within the palette list
- **scenePaletteList**: This determines whether to check the scene palette list or the element palette list

getPaletteld

Description

This function returns the id of the palette in the current palette list in the ColourView.

Syntax

```
PaletteManager.getPaletteId(int index);
```

Arguments

- **index**: This is the index of palettes within the palette list

getPaletteld

Description

This function returns the id of the palette in the current palette list in the ColourView.

Syntax

```
PaletteManager.getPaletteId(int index, scenePaletteList );
```

Arguments

- **index**: This is the index of palettes within the palette list
- **scenePaletteList**: This determines whether to check the scene palette list or the element palette list

PenstyleManager

This set of functions is used to query/modify the current penstyle and list of penstyles. The list of penstyles includes the brush, pencil and texture styles.

The following is a list of the PenstyleManager functions:

- `getNumberOfPenstyles`, on page 97
- `getPenstyleName`, on page 97
- `setCurrentPenstyleByName`, on page 97
- `setCurrentPenstyleByIndex`, on page 97
- `changeCurrentPenstyleMinimumSize`, on page 98
- `changeCurrentPenstyleMaximumSize`, on page 98
- `getCurrentPenstyleIndex`, on page 99
- `changeCurrentPenstyleOutlineSmoothness`, on page 98
- `changeCurrentPenstyleCenterlineSmoothness`, on page 98
- `changeCurrentPenstyleEraserFlag`, on page 98
- `getCurrentPenstyleMinimumSize`, on page 99
- `getCurrentPenstyleMaximumSize`, on page 99
- `getCurrentPenstyleOutlineSmoothness`, on page 99
- `getCurrentPenstyleCenterlineSmoothness`, on page 99
- `getCurrentPenstyleEraserFlag`, on page 100
- `exportPenstyleToString`, on page 100
- `exportPenstyleListToString`, on page 100
- `importPenstyleListFromString`, on page 100
- `savePenstyles`, on page 100

Example

```
function queryPenstyles()
{

    var num = PenstyleManager.getNumberOfPenstyles();
    for ( var i =0 ; i < num ; ++i )
    {
        System.println( "penstyle name is " + PenstyleManager.getPenstyleName(i) );
    }

    System.println("The current penstyle has min size of " +
        PenstyleManager.getCurrentPenstyleMinimumSize() +
        " and maximum size of " +
        PenstyleManager.getCurrentPenstyleMaximumSize());
}
```

getNumberOfPenstyles

Description

This function returns the number of penstyles.

Syntax

```
PenstyleManager.getNumberOfPenstyles();
```

Arguments

None

getPenstyleName

Description

This function returns the name of the penstyle.

Syntax

```
PenstyleManager.getPenstyleName(int index);
```

Arguments

- **index**: index of style within the list

getCurrentPenstyleName

Description

This function returns the name of the current pen style.

Syntax

```
PenstyleManager.getCurrentPenstyleName();
```

Arguments

None

setCurrentPenstyleByName

Description

This function sets the current penstyle by name.

Syntax

```
PenstyleManager.setCurrentPenstyleByName("name");
```

Arguments

- **name**: This is the name of penstyle

setCurrentPenstyleByIndex

Description

This function sets the current penstyle

Syntax

```
PenstyleManager.setCurrentPenstyleByIndex(int index);
```

Arguments

- **index**: This is the penstyle index

changeCurrentPenstyleMinimumSize

Description

This function sets the current penstyle minimum size.

Syntax

```
PenstyleManager.changeCurrentPenstyleMinimumSize
```

Arguments

- **minimum:** This is the new minimum size

changeCurrentPenstyleMaximumSize

Description

This function sets the current penstyle maximum size.

Syntax

```
PenstyleManager.changeCurrentPenstyleMaximumSize(double maximum);
```

Arguments

- **maximum:** This is the new maximum size

changeCurrentPenstyleOutlineSmoothness

Description

This function sets the current penstyle outline smoothness.

Syntax

```
PenstyleManager.changeCurrentPenstyleOutlineSmoothness(int smooth);
```

Arguments

- **smooth:** This is the new smoothness value

changeCurrentPenstyleCenterlineSmoothness

Description

This function sets the current penstyle centreline smoothness.

Syntax

```
PenstyleManager.changeCurrentPenstyleCenterlineSmoothness(int smooth);
```

Arguments

- **smooth:** This is the new smoothness value

changeCurrentPenstyleEraserFlag

Description

This function sets the current penstyle eraser flag.

Syntax

```
PenstyleManager.changeCurrentPenstyleEraserFlag(flag);
```

Arguments

- **flag:** This is the eraser setting

getCurrentPenstyleIndex

Description

This function gets the index of the current penstyle.

Syntax

```
PenstyleManager.getCurrentPenstyleIndex();
```

Arguments

None

getCurrentPenstyleMinimumSize

Description

This function gets the current penstyle minimum size.

Syntax

```
PenstyleManager.getCurrentPenstyleMinimumSize();
```

Arguments

None

getCurrentPenstyleMaximumSize

Description

This function gets the current penstyle maximum size.

Syntax

```
PenstyleManager.getCurrentPenstyleMaximumSize();
```

Arguments

None

getCurrentPenstyleOutlineSmoothness

Description

This function gets the current penstyle outline smoothness.

Syntax

```
PenstyleManager.getCurrentPenstyleOutlineSmoothness();
```

Arguments

None

getCurrentPenstyleCenterlineSmoothness

Description

This function gets the current penstyle center line smoothness.

Syntax

```
PenstyleManager.getCurrentPenstyleCenterlineSmoothness();
```

Arguments

None

getCurrentPenstyleEraserFlag

Description

This function gets the current penstyle eraser flag.

Syntax

```
PenstyleManager.getCurrentPenstyleEraserFlag();
```

Arguments

None

exportPenstyleToString

Description

This function creates a string representing the penstyle which can be used to store the penstyle and import it later.

Syntax

```
PenstyleManager.exportPenstyleToString(int index);
```

Arguments

- **index**: This is the penstyle index

exportPenstyleListToString

Description

This function formats the penstyle list into a string, which can be used to store the penstyle list and import it later.

Syntax

```
PenstyleManager.exportPenstyleListToString();
```

Arguments

None

importPenstyleListFromString

Description

This function imports a penstyle list from a previously formatted penstyle string.

Syntax

```
PenstyleManager.importPenstyleListFromString("str");
```

Arguments

- **str**: This is the formatted penstyle list (created from a previous call to exportPenstyleToString or exportPenstyleListToString).

savePenstyles

Description

This function saves the pen styles.

Syntax

```
PenstyleManager.savePenstyles();
```

Arguments

None

Preferences

With the Preferences functions, you can retrieve information about the whole preference system. The user can set and retrieve the value of any preferences in the software.

The actual name and current value of a preference are in stored in the user configuration file.

The file `prefs.xml` contains a description of all preferences recognized by the application. The keyword to access each predefined preference is also found in that file.

Scripts can change or retrieve any existing preference, and may create new preferences.

The following is a list of the Preferences functions:

- `getBool`, on page 101
- `getColor`, on page 102
- `getDouble`, on page 102
- `getInt`, on page 102
- `getString`, on page 102
- `setBool`, on page 102
- `setColor`, on page 103
- `setDouble`, on page 103
- `setInt`, on page 103
- `setString`, on page 103

Example

This script that will toggle the property to automatically save layout on exit.

```
function toggleAutoSaveLayout()
{
    var b;
    b = Application.preferences.getBool( "AUTO_SAVE_LAYOUT", false );
    print( "preference to automatically save the layout was " + b );
    Application.preferences.setBool( "AUTO_SAVE_LAYOUT", !b );
    b = Application.preferences.getBool( "AUTO_SAVE_LAYOUT", false );
    print( "preference for auto save layout is now " + b );
}
```

getBool

Description

This function returns the current value of a boolean preference.

Syntax

```
preferences.getBool( keyword, defaultValue );
```

Arguments

- **keyword**: The preference keyword.
- **defaultValue**: This is the value that will be returned when the preference has not been set.

getColor

Description

This function returns the current value of a Colour preference.

Syntax

```
preferences.getColor( keyword, defaultValue );
```

Arguments

- **Keyword:** The preference keyword.
- **defaultValue:** This is the value that will be returned when the preference has not been set.

getDouble

Description

This function returns the current value of a double (floating point) preference

Syntax

```
preferences.getDouble( keyword, defaultValue );
```

Arguments

- **Keyword:** The preference keyword.
- **defaultValue:** This is the value that will be returned when the preference has not been set.

getInt

Description

This function returns the current value of an integer preference

Syntax

```
preferences.getInt( keyword, defaultValue );
```

Arguments

- **Keyword:** The preference keyword.
- **defaultValue:** This is the value that will be returned when the preference has not been set.

getString

Description

This function gets the value of a string preference.

Syntax

```
preferences.getString( name, defaultValue );
```

Arguments

- **defaultValue:** The value that is returned when the preference is not set.

setBool

Description

This function sets the value of a boolean preference.

Syntax

```
preferences.setBool( keyword, value );
```

Arguments

- **Keyword:** The preference keyword.
- **Value:** This is the new preference value.

setColor

Description

This function sets the value of a colour preference

Syntax

```
preferences.setColor( keyword, value );
```

Arguments

- **Keyword:** The preference keyword.
- **Value:** This is the new preference value.

setDouble

Description

This function sets the value of a double (floating point) preference.

Syntax

```
preferences.setDouble( keyword, value );
```

Arguments

- **Keyword:** The preference keyword.
- **Value:** This is the new preference value.

setInt

Description

This function sets the value of a integer preference.

Syntax

```
preferences.setInt( keyword, value )
```

Arguments

- **Keyword:** The preference keyword.
- **Value:** This is the new preference value.

setString

Description

This function sets the value of a string preference.

Syntax

```
preferences.setString( name, value );
```

Arguments

- **name:** the preference keyword
- **Value:** The value that is returned when the preference is not set.

Render

The Render class is used to render the scene or a part of the scene. The scripting environment can receive notifications when scene frame is ready.

The following is a list of the Preferences functions:

- `frameReady`, on page 105
- `renderFinished`, on page 105
- `setCombine`, on page 105
- `setFieldType`, on page 105
- `setBgColor`, on page 106
- `setResolution`, on page 106
- `setRenderDisplay`, on page 106
- `setWriteEnabled`, on page 106
- `renderScene`, on page 106
- `renderSceneAll`, on page 107
- `cancelRender`, on page 107

Example

```
class RenderHandler
{
    function frameReady(frame, frameCel)
    {
        // Store the input rendered frame in the export directory.
        var pos = frame.toString();

        for(var i = pos.length; i < 5; i++)
            pos = "0" + pos;

        var exportFile = exporter.getExportDir() + "frame" + pos + ".tga";
        frameCel.imageFile(exportFile);
    }

    function renderFinished()
    {
        var exportDir = new Dir(exporter.getExportDir());
        var renderedFiles = exportDir.entryList('*.tga');

        MessageBox.information(renderedFiles.length + " frames rendered to: " +
            exportDir.path);
    }
}

function renderFrames()
{
    // Render the first 10 frames of the scene.
    var handler = new RenderHandler;

    connect(render, "frameReady(int,SM_CelWrapper&)", handler.frameReady);
}
```

```

connect(render, "renderFinished()", handler.renderFinished);

render.setRenderDisplay("Display");
render.renderScene(1, 10);

disconnect(render, "frameReady(int, SM_CelWrapper&)", handler.frameReady);
disconnect(render, "renderFinished()", handler.renderFinished);
}

```

frameReady

Description

Event that notifies the script that a certain frame is available and at which location.

Syntax

```
render.frameReady(int frame, SM_CelWrapper &frameCel);
```

Arguments

- **frame**: This is the rendered frame number
- **frameCel**: This is the rendered frame cel

renderFinished

Description

Event that notifies the script when the render has completed.

Syntax

```
render.renderFinished();
```

Arguments

None

setCombine

Description

Set if rendered frames sets should be combined and in which order.

Specify these options if you are rendering in PAL or NTSC format.

Syntax

```
setCombine(autoCombine, secondFieldFirst);
```

Arguments

- **autoCombine**: This automatically combines the two rendered frame field sets
- **secondFieldFirst**: This inserts the second frame field set at the beginning

setFieldType

Description

Sets the frame output format.

Syntax

```
render.setFieldType(int type);
```

Arguments

- **type**: This is the frame output format: 0 - None, 1 - NTSC, 2 - PAL

setBgColor

Description

Set the background color to use when rendering in scene machine mode.

Syntax

```
render.setBgColor(QColor bgColor);
```

Arguments

- **bgColor**: This is the background colour

setResolution

Description

Set the scene resolution to use for rendering.

Syntax

```
render.setResolution(int x, int y);
```

Arguments

- **x**: This is the width in pixels
- **y**: This is the height in pixels

setRenderDisplay

Description

Set which display module to use for rendering. "Display All" uses the global unconnected display module.

Syntax

```
render.setRenderDisplay("name");
```

Arguments

- **name**: This is the display name

setWriteEnabled

Description

Enable or disable write modules during the render.

Syntax

```
render.setWriteEnabled(enabled);
```

Arguments

- **enable**: This enables or disables the write modules

renderScene

Description

Render a part of the scene.

Syntax

```
render.renderScene(int fromFrame, int toFrame);
```

Arguments

- **fromFrame**: This is the render start frame

renderSceneAll

Description

Render the complete scene.

Syntax

```
render.renderSceneAll();
```

Arguments

None

cancelRender

Description

Interrupt an active render.

Syntax

```
render.cancelRender();
```

Arguments

None

Scene

With the Scene functions, you can retrieve and set global scene attributes, like the aspect ratio of the cells in the scene grid.

The following is a list of the Scene functions:

- `beginUndoRedoAccum`, on page 109
- `cancelUndoRedoAccum`, on page 109
- `clearHistory`, on page 109
- `coordAtCenterX`, on page 110
- `coordAtCenterY`, on page 110
- `currentEnvironment`, on page 110
- `currentJob`, on page 110
- `currentProjectPath`, on page 110
- `currentProjectPathRemapped`, on page 111
- `currentResolutionX`, on page 111
- `currentResolutionY`, on page 111
- `currentScene`, on page 111
- `currentVersion`, on page 111
- `defaultResolutionFOV`, on page 112
- `defaultResolutionName`, on page 112
- `defaultResolutionX`, on page 112
- `defaultResolutionY`, on page 112
- `endUndoRedoAccum`, on page 112
- `fromOGL`, on page 113
- `getCameraMatrix`, on page 113
- `getFrameRate`, on page 113
- `numberOfUnitsX`, on page 113
- `numberOfUnitsY`, on page 113
- `numberOfUnitsZ`, on page 114
- `saveAll`, on page 114
- `saveAsNewVersion`, on page 114
- `setCoordAtCenter`, on page 114
- `setNumberOfUnits`, on page 114
- `setUnitsAspectRatio`, on page 115
- `unitsAspectRatioX`, on page 115
- `unitsAspectRatioY`, on page 115
- `setDefaultResolution`, on page 115
- `setFrameRate`, on page 116
- `toOGL`, on page 115
- `unitsAspectRatioX`, on page 115
- `unitsAspectRatioY`, on page 115
- `setDefaultResolution`, on page 115
- `setFrameRate`, on page 116

Example

This script uses an undo/redo wrapper to enclose several functions in one command called "Set Scene". The Set Scene command sets the aspect ratio of the scene, sets the number of units of the scene and sets the centre coordinates.

```
function setScene()
{
    // Sets the beginning of the undo/redo command wrapper
    scene.beginUndoRedoAccum("Set Scene");
    // Sets the aspect ratio of the scene to 4, 3
    scene.setUnitsAspectRatio(4, 3);
    // Sets the number of units in the scene
    scene.setNumberOfUnits(1200, 900, 12);
    // Sets the value of the center coordinate
    scene.setCoordAtCenter(5000, 5000);
    // Terminates the undo/redo command wrapper
    scene.endUndoRedoAccum();
}
```

beginUndoRedoAccum

Description

This function starts the accumulation of all of the functions between it and the `endUndoRedoAccum` function as one command that will appear in the undo/redo list. If you do not use this function with `endUndoRedoAccum`, each function in the script generates a separate undo/redo entry.

Syntax

```
scene.beginUndoRedoAccum("commandname");
```

Arguments

"commandname"

- "commandname": The name of the command to be added to the undo/redo list.

cancelUndoRedoAccum

Description

This function cancels the accumulation of undo/redo commands. No command will be added to the undo/redo list and all commands that have already been executed will be rolled-back (undone).

This function can only be called after a call to `beginUndoRedoAccum()`.

Syntax

```
scene.cancelUndoRedoAccum();
```

Arguments

None

clearHistory

Description

This function clears the command history. After this call it is not possible to undo the command.

Syntax

```
scene.clearHistory();
```

Arguments

None.

coordAtCenterX

Description

This function returns the X value of the centre coordinate of the scene grid.

Syntax

```
scene.coordAtCenterX();
```

Arguments

None.

coordAtCenterY

Description

This function returns the Y value of the centre coordinate of the scene grid.

Syntax

```
scene.coordAtCenterY();
```

Arguments

None.

currentEnvironment

Description

This function returns the name of the current environment. This command only applies to Animate.

Syntax

```
scene.currentEnvironment();
```

Arguments

None

currentJob

Description

This function returns the name of the current job. This command only works with Animate.

Syntax

```
scene.currentJob();
```

Arguments

None

currentProjectPath

Description

This function returns the path of the current project or the current scene. For Animate, on Windows, the path is untranslated (e.g. /USA_DB/jobs/j/scene-a).

Syntax

```
scene.currentProjectPath();
```

Arguments

None

currentProjectPathRemapped

Description

This function returns the path of the current project or the current scene. The path returned is translated to contain the actual physical location. (e.g: C:\usadata000\jobs\j\scene-a)

Syntax

```
scene.currentProjectPathRemapped();
```

Arguments

None

currentResolutionX

Description

This function returns the current preview resolution. For example, when the current resolution is 720x540 pixels this function will return 720.

Syntax

```
scene.currentResolutionX();
```

Arguments

None

currentResolutionY

Description

This function returns the current preview resolution. For example, when the current resolution is 720x540 pixels this function will return 540.

Syntax

```
scene.currentResolutionY();
```

Arguments

None

currentScene

Description

This function returns the name of the current scene.

Syntax

```
scene.currentJob();
```

Arguments

None.

currentVersion

Description

This function returns the name or the number of the current version. In Animate, a version is always a number, starting at 1. In Digital Pro, it is possible to have named versions.

Syntax

```
scene.currentVersion();
```

Arguments

None

defaultResolutionFOV

Description

This function returns the default resolution field of view (FOV). The default FOV is a global scene parameter.

Syntax

```
scene.defaultResolutionFOV();
```

Arguments

None

defaultResolutionName

Description

This function returns the default resolution name. The resolution name is a global parameter saved with the project. It may be empty when the project is used as a custom resolution, which is not one of the pre-defined resolutions.

Syntax

```
scene.currentResolutionName();
```

Arguments

None

defaultResolutionX

Description

This function returns the default resolution. This resolution is a global parameter saved with the project, not the current preview resolution. For example, when the default scene resolution is 720x540 pixels this function will return 720.

Syntax

```
scene.defaultResolutionX();
```

Arguments

None

defaultResolutionY

Description

This function returns the default resolution. This resolution is a global parameter saved with the project, not the current preview resolution. For example, when the default scene resolution is 720x540 pixels this function will return 540.

Syntax

```
scene.defaultResolutionY();
```

Arguments

None

endUndoRedoAccum

Description

This function ends the accumulation all of the functions between it and the `beginUndoRedoAccum` function as one command that will appear in the undo/redo list. If you do not use this function with `beginUndoRedoAccum`, each function in the script generates a separate undo/redo entry.

Syntax

```
scene.endUndoRedoAccum();
```

Arguments

None.

getFrameRate

Description

This function returns the frame rate, as frame per seconds.

Syntax

```
scene.getFrameRate();
```

Arguments

None

fromOGL

Description

This function converts an OGL coordinate into a field coordinate.

Syntax

```
scene.fromOGL(pointOrVector);
```

Arguments

`pointOrVector`: can be either a2D point or a3D point or a vector object see the script module for details.

getCameraMatrix

Description

This function returns the model matrix for the default camera.

Syntax

```
scene.getCameraMatrix(frame);
```

Arguments

`frame`: is the int frame number.

numberOfUnitsX

Description

This function returns the number of units in the X axis of the scene grid.

Syntax

```
scene.numberOfUnitsX();
```

Arguments

None.

numberOfUnitsY

Description

This function returns the number of units in the Y axis of the scene grid.

Syntax

```
scene.numberOfUnitsY();
```

Arguments

None.

numberOfUnitsZ

Description

This function returns the number of units in the Z-axis of the scene grid.

Syntax

```
scene.numberOfUnitsZ();
```

Arguments

None.

saveAll

Description

This function performs the " save all " command. Effectively, this saves the entire project and all modified files.

Syntax

```
scene.saveAll();
```

Arguments

None

saveAsNewVersion

Description

This function saves the project as a new version.

Syntax

```
scene.saveAsNewVersion( name , markAsDefault )
```

Arguments

- **name**: The name of the version. Animate requires that the name be a number.
- **markAsDefault**: This is boolean to indicate to mark this version as the default version. This field is only used by Animate.

setCoordAtCenter

Description

This functions sets the value of the centre (X, Y) coordinates.

Syntax

```
scene.setCoordAtCenter(x, y);
```

Arguments

x, y

- **x, y**: The value of the X and Y coordinate at the centre of the grid.

setNumberOfUnits

Description

This function sets the number of X, Y, and Z units in the scene grid.

Syntax

```
scene.setNumberOfUnits(x, y, z);
```

Arguments

x, y, z

- **x, y, z**: The X, Y and Z values of the scene grid.

setUnitsAspectRatio

Description

This function sets the aspect ratio of the scene. The scene's final aspect ratio will be:

$$X * \text{numberOfUnitsX}() / Y * \text{numberOfUnitsY}()$$

Syntax

```
scene.setUnitsAspectRatio(x, y);
```

Arguments

x, y

- **x, y**: The X, Y value of the new aspect ratio.

toOGL

Description

This function converts a field coordinate into an OGL coordinate.

Syntax

```
scene.toOGL(pointOrVector);
```

Arguments

pointOrVector: can be either a 2D point or a 3D point or a vector object. See the script module for details.

unitsAspectRatioX

Description

This function returns the X value of the aspect ratio of the cells in the scene grid.

Syntax

```
scene.unitsAspectRatioX();
```

Arguments

None.

unitsAspectRatioY

Description

This function returns the Y value of the aspect ratio of the cells in the scene grid.

Syntax

```
scene.unitsAspectRatioY();
```

Arguments

None.

setDefaultResolution

Description

This function allows the default scene resolution and field of view to be changed.

Syntax

```
scene.setDefaultResolution( X, Y, fov );
```

Arguments

- **x, y**: Set the X and Y resolution for the scene in pixels.
- **fov**: Set the field of view in degree. Typical value is 41.112.

setFrameRate

Description

This function allows the default frame rate of the project to be changed. The frame rate is expressed as frame per second. Typical value is 12, 24 or 30.

Syntax

```
scene.setFrameRate( fps );
```

Arguments

- `fps`: The frame rate.

Selection

With the Selection functions, you can retrieve information about the modules or columns you have selected within a view. These functions work best if run from an icon in the Views Toolbar

Following are the Selection functions:

- `clearSelection`, on page 118
- `addDrawingColumnToSelection`, on page 118
- `addColumnToSelection`, on page 118
- `addNodeToSelection`, on page 118
- `extendSelectionWithColumn`, on page 118
- `numberOfCellColumnsSelected`, on page 119
- `numberOfFramesSelected`, on page 119
- `numberOfNodesSelected`, on page 119
- `selectAll`, on page 119
- `selectedCellColumn`, on page 119
- `selectedNode`, on page 120
- `setSelectionFrameRange`, on page 120

Example

This script prints the name of the selected columns and their frame values to the shell.

```
function dumpSelectedColumn()
{
/* The numberOfCellColumnsSelected function loops to check all of the selected
columns and determine their frame values.

The selectedCellColumn function prints the name of the selected column to the shell
and then loops through the values in the column to print them as well. */
var ncol = selection.numberOfCellColumnsSelected();
var nframe = frame.numberOf();
var i;
for (i = 0; i < ncol; ++i)
{
var f;
var c = selection.selectedCellColumn(i);
System.println("Column " + c);
for (f = 1; f <= nframe; ++f)
{
var value = column.getEntry(c, 1, f);
var keyframe = column.isKeyFrame(c, 1, f);
if (keyframe) value += " (keyframe)";
System.println(" " + f + ": " + value);
}
}
}
```

clearSelection

Description

This function clears the selection.

Syntax

```
selection.clearSelection();
```

Arguments

None.

addDrawingColumnToSelection

Description

This function adds the drawing column and it's associated read node to the selection.

Syntax

```
selection.addDrawingColumnToSelection(columnName);
```

Arguments

- `columnName`: name of column.

addColumnToSelection

Description

This function adds a column to the selection.

Syntax

```
selection.addColumnToSelection(column);
```

Arguments

- `column name`: name of column.

addNodeToSelection

Description

This function adds a node to the selection.

Syntax

```
selection.addNodeToSelection(node);
```

Arguments

- `node`: name of node.

extendSelectionWithColumn

Description

This function adds the drawing column to the selection.

If the column is a drawing column, also adds the associated read node to the selection.

Syntax

```
selection.extendSelectionWithColumn(columnName);
```

Arguments

- `columnName`: name of column.

numberOfCellColumnsSelected

Description

To be used in the Xsheet view context.

This function returns a value for the number of selected columns.

Syntax

```
selection.numberOfCellColumnsSelected();
```

Arguments

None.

numberOfFramesSelected

Description

This function returns the number of frames -selected (to be used in the xhseetview only)

Syntax

```
selection.numberOfFramesSelected();
```

Arguments

None

numberOfNodesSelected

Description

This function returns the number of modules that are selected.

Syntax

```
selection.numberOfNodesSelected();
```

Arguments

None.

selectAll

Description

This function selects all nodes and all columns in the scene.

Syntax

```
selection.selectAll();
```

Arguments

None

selectedCellColumn

Description

To be used in the Xsheet view context.

This function returns the name of the selected column.

Syntax

```
selection.selectedCellColumn(int i);
```

Arguments

int i

- **int i**: The index value of each selected column. The value must be between 0 and the `numberOfColumnsSelected` function.

selectedNode

Description

This function returns the path of the selected node.

Syntax

```
selection.selectedNode(int i);
```

Arguments

`int i`

- `int i`: The index value of each selected node. The value must be between 0 and the `numberOfNodesSelected` function.

setSelectionFrameRange

Description

This function sets the frame range for the selection.

Syntax

```
selection.setSelectionFrameRange(int start, int end );
```

Arguments

- `int start`: The start frame of the selection range.
- `int end`: The end frame of the selection range.

Sound

The Sound class is used to access the scene's soundtrack in part or in whole. The scripting environment can receive notifications when scene frame is ready.

The following is a list of the Sound functions:

- `setSampleRate`, on page 121
- `setChannelSize`, on page 121
- `setChannelCount`, on page 121
- `getSoundtrack`, on page 122
- `getSoundtrackAll`, on page 122

Example

```
// Retrieves a 16 bit 48 KHz stereo soundtrack of frames 50 to 100.
sound.setSampleRate(48000);
sound.setChannelSize(16);
sound.setChannelCount(2);
var soundFile = sound.getSoundtrack(50, 100);
MessageBox.information("Soundtrack file location: " + soundFile.path());
```

setSampleRate

Description

Sets the audio sample rate in Hz (i.e. 22050, 44100, ...)

Syntax

```
sound.setSampleRate(double rate);
```

Arguments

- `rate`: This is the audio sample rate

setChannelSize

Description

Sets the audio channel size (i.e. 8 or 16 bit).

Syntax

```
sound.setChannelSize(int size);
```

Arguments

- `size`: This is the audio channel size

setChannelCount

Description

Sets the number of audio channels (i.e 1 for mono and 2 for stereo).

Syntax

```
sound.setChannelCount(int count);
```

Arguments

- `size`: This is the audio channel count.

getSoundtrack

Description

Returns a part of the scene's soundtrack in a temporary file in WAV format.

Syntax

```
sound.getSoundtrack(int fromFrame, int toFrame);
```

Arguments

- **fromFrame**: This is the soundtrack start frame.
- **toFrame**: This is the soundtrack end frame.

getSoundtrackAll

Description

Returns the scene's soundtrack in a temporary file in WAV format.

Syntax

```
sound.getSoundtrackAll();
```

Arguments

None

SpecialFolders

With the SpecialFolders functions, you can retrieve information about the different folders (directories) used by the application. All of the functions are read-only. They return strings that represent folders in use by the various applications. Depending on the application (e.g. Toon Boom Animate versus Toon Boom Digital Pro), the same content is stored in a different location.

The following is a list of the SpecialFolders functions:

- app, on page 123
- bin, on page 124
- config, on page 124
- etc, on page 124
- lang, on page 124
- library, on page 125
- platform, on page 125
- plugins, on page 125
- resource, on page 125
- root, on page 126
- temp, on page 126
- userConfig, on page 126

Example

```
function specialFolder()
{
    print( "Special Folders" );
    print( "root:: " + Application.specialFolders.root );
    print( "config: " + Application.specialFolders.config );
    print( "resource: " + Application.specialFolders.resource );
    print( "etc: " + Application.specialFolders.etc );
    print( "lang: " + Application.specialFolders.lang );
    print( "platform " + Application.specialFolders.platform );
    print( "app " + Application.specialFolders.app );
    print( "bin: " + Application.specialFolders.bin );
    print( "library: " + Application.specialFolders.library );
    print( "plugins: " + Application.specialFolders.plugins );
    print( "temp: " + Application.specialFolders.temp );
    print( "userConfig: " + Application.specialFolders.userConfig );
}
```

app

Description

A read-only property containing the folder where the platforms specific applications are stored. Application and Binary folders are different on OS X, but are identical on all other platforms.

Syntax

```
specialFolders.app
```

Arguments

None

bin

Description

This is a read-only property that contains the folder where the platforms specific binaries are stored. Application and Binary folders are different on OS X. They are identical on all other platforms.

Syntax

```
specialFolders.bin
```

Arguments

None

config

Description

This is a read-only property that contains the folder where application configuration files are stored. Normally, this is the /etc folder.

Syntax

```
specialFolders.config
```

Arguments

None

etc

Description

This is a read-only property that indicates where the <install>/etc folder is.

Syntax

```
specialFolders.etc;
```

Arguments

None

htmlHelp

Description

This is a read-only property that contains the folder where the html help folder is.

Syntax

```
about.htmlHelp;
```

Arguments

None

lang

Description

This is a read-only property that contains the folder where the language files are stored.

Syntax

```
specialFolders.lang;
```

Arguments

None

library

Description

This is a read-only property that contains the folder where the platforms specific libraries are stored.

Syntax

```
specialFolders.library;
```

Arguments

None

pdf

Description

This is a read-only property that contains the folder where the pdf help folder is.

Syntax

```
about.pdf;
```

Arguments

None

platform

Description

This is a read-only property that contains the platform specific folder.

Syntax

```
specialFolders.platform ;
```

Arguments

None

plugins

Description

This is a read-only property that contains where the platform specific plugins are stored.

Syntax

```
specialFolders.plugins
```

Arguments

None

resource

Description

This is a read-only property that contains where the resources files are stored.

Syntax

```
specialFolders.resource
```

Arguments

None

root

Description

This is a read-only property for the root installation folder.

Syntax

```
specialFolders.root
```

Arguments

None

temp

Description

This is a read-only property that contains where the application will create its temporary files.

Syntax

```
specialFolders.temp
```

Arguments

None

userConfig

Description

This is a read-only property that contains the folder where the user configuration is stored.

Syntax

```
specialFolders.userConfig
```

Arguments

None

Timeline

With the Timeline functions, you can return values for layers and frames in the Timeline window.

There are two main groups of Timeline functions:

- **selToXX**: the selection functions work with a selection using the selIdx parameter.
- **layerToXX**: the layer functions work with Timeline layers using the layerIdx parameter.

The **layerIdx** and **selIdx** parameters are used in many of the functions:

- ⇒ **layerIdx**: this is an integer that represents the layer in the Timeline. The first layer in the Timeline is considered 0 (zero).
- ⇒ **selIdx**: this is an integer that represents the selection in the Timeline. It is a number from 0 to the number of layers selected minus 1.

Following are the Timeline functions:

- `firstFrameSel`, on page 129
- `isAncestorOf`, on page 129
- `layerIsColumn`, on page 129
- `layerIsNode`, on page 129
- `layerToColumn`, on page 130
- `layerToNode`, on page 130
- `numFrameSel`, on page 130
- `numLayers`, on page 130
- `numLayerSel`, on page 130
- `parentNodeIndex`, on page 131
- `sellsColumn`, on page 131
- `sellsNode`, on page 131
- `selToColumn`, on page 131
- `selToLayer`, on page 132
- `selToNode`, on page 132
- `setDisplayToUnconnected`, on page 132

Example

```
function printTimelineLayerInformation()
{
    //
    // Print out information for all layers
    //

    var numLayers = Timeline.numLayers;

    for ( var i = 0; i < numLayers;i++ )
    {
        if ( Timeline.layerIsNode( i ) )
            System.println(i + ": is a node named " + Timeline.layerToNode(i));

        if ( Timeline.layerIsColumn(i ) )
            System.println( " " + i + ": also has an xsheet column named " +
Timeline.layerToColumn(i ) );
    }

    //
    // Print out information on selected layers
    //

    System.println("number of frames selected is " + Timeline.numFrameSel + "
starting at " + Timeline.firstFrameSel);

    var numSelLayers = Timeline.numLayerSel;

    for ( var i = 0; i < numSelLayers; i++ )
    {
        if ( Timeline.selIsNode( i ) )
            System.println(" " + i + ": is a SELECTED node layer with name " +
Timeline.selToNode(i));

        if ( Timeline.selIsColumn(i ) )
            System.println( i + ": SELECTED layer name is " + Timeline.selToColumn(i ) );
    }
}
```

firstFrameSel

Description

This function returns the number of the first frame in the Timeline selection or the current frame, if only one frame is selected.

Syntax

```
Timeline.firstFrameSel;
```

Arguments

None.

isAncestorOf

Description

This function returns true or false to identify if a layer is the parent of another layer.

Syntax

```
Timeline.isAncestorOf(parentLayerIdx, layerIdx);
```

Arguments

parentLayerIdx, **layerIdx**

- **parentLayerIdx**: This is an integer that represents a layer that has nested sub-layers (children) in the Timeline.
- **layerIdx**: This is an integer that represents a layer in the Timeline.

layerIsColumn

Description

This function returns true or false to identify if the Timeline layer is linked to a column in the Xsheet.

Syntax

```
Timeline.layerIsColumn(layerIdx);
```

Arguments

layerIdx

- **layerIdx**: This is an integer that represents the layer in the Timeline.

layerIsNode

Description

This function returns true or false to identify if the Timeline layer is linked to a module (node) in the Network.

Syntax

```
Timeline.layerIsNode(layerIdx);
```

Arguments

layerIdx

- **layerIdx**: This is an integer that represents the layer in the Timeline.

layerToColumn

Description

This function returns the column name for the Timeline layer. It returns an empty string if the layer is not a column.

Syntax

```
Timeline.layerToColumn(layerIdx);
```

Arguments

layerIdx

- **layerIdx**: This is an integer that represents the layer in the Timeline.

layerToNode

Description

This function returns the node (module) index from the Network for the Timeline layer. It returns an empty string if the layer is not a node.

Syntax

```
Timeline.layerToNode(layerIdx);
```

Arguments

layerIdx

- **layerIdx**: This is an integer that represents the layer in the Timeline.

numFrameSel

Description

This function returns the number of the selected frame in the Timeline, if only one frame is selected. It will return zero (0) if no frames are selected.

Syntax

```
Timeline.numFrameSel;
```

Arguments

None.

numLayers

Description

This function returns the number of layers in the Timeline.

Syntax

```
Timeline.numLayers;
```

Arguments

None.

numLayerSel

Description

This function returns the number of layers that are selected in the Timeline.

A peg layer counts for six layers, seven if the Scale properties are set to separate functions in the Peg Module Editor.

Syntax

```
Timeline.numLayerSel;
```

Arguments

None.

parentNodeIndex

Description

This function returns a layer identifier (`layerIdx`) for the parent of the layer (`layerIdx`). Peg Modules are often used to create parent-child hierarchies that control the animation of several related modules.

Syntax

```
Timeline.parentNodeIndex(layerIdx);
```

Arguments

layerIdx

- `layerIdx`: This is an integer that represents the layer in the Timeline.

sellsColumn

Description

This function returns true or false to indicate if the Timeline selection has a column in the Xsheet.

Syntax

```
Timeline.selIsColumn(selIdx);
```

Arguments

selIdx

- `selIdx`: This is an integer that represents the selection in the Timeline. It is a number from 0 to the `numLayerSel - 1`.

sellsNode

Description

This function returns true or false to identify if the Timeline selection is linked to a module (node) in the Network.

Syntax

```
Timeline.selIsNode(selIdx);
```

Arguments

selIdx

- `selIdx`: This is an integer that represents the selection in the Timeline. It is a number from 0 to the `numLayerSel - 1`.

selToColumn

Description

This function returns the column name for the Timeline selection.

Syntax

```
Timeline.selToColumn(selIdx);
```

Arguments

selIdx

- `selIdx`: This is an integer that represents the selection in the Timeline. It is a number from 0 to the `numLayerSel - 1`.

selToLayer

Description

This function returns the layer identifier (`layerIdx`) for the selection (`selIdx`) in the Timeline.

Syntax

```
Timeline.selToLayer(selIdx);
```

Arguments

`selIdx`

- `selIdx`: This is an integer that represents the selection in the Timeline. It is a number from 0 to the `numLayerSel - 1`.

selToNode

Description

This function returns the node name from the module in the Network for the Timeline selection.

Syntax

```
Timeline.selToNode(selIdx);
```

Arguments

`selIdx`

- `selIdx`: This is an integer that represents the selection in the Timeline. It is a number from 0 to the `numLayerSel - 1`.

setDisplayToUnconnected

Description

This function sets the Display to Unconnected in the Timeline. It returns false if it was unable to set the Display.

Syntax

```
Timeline.setDisplayToUnconnected();
```

Arguments

None

View

The View functions provide information about the contents of selected View windows.

The following is a list of the View functions:

- `column`, on page 133
- `currentView`, on page 133
- `group`, on page 133
- `refreshViews`, on page 134
- `type`, on page 134

Example

This script prints the name of the View type in the Shell that started the Stage Module.

```
function viewScript()  
{  
    var myView = view.currentView();  
    System.println(view.type(myView));  
}
```

column

Description

This function returns the name of the column for the currently displayed function in the Function View.

Syntax

```
view.column(currentView);
```

Arguments

`currentView`

- `currentView`: The current view value, as returned by the `currentView` function.

currentView

Description

This function returns a unique identifier for the current, active View.

Syntax

```
view.currentView();
```

Arguments

None.

group

Description

This function returns the name of the current Group Module in the active Network View.

Syntax

```
view.group(currentView);
```

Arguments

`currentView`

- `currentView`: The current view value, as returned by the `currentView` function.

refreshViews

Description

This function forces a refresh of the drawing and scene planning views.

Syntax

```
view.refreshViews();
```

Arguments

None

type

Description

This function returns a string that indicates what type of View the `currentView` is.

Syntax

```
view.type(currentView);
```

Arguments

`currentView`

- `currentView`: The current view value, as returned by the `currentView` function.

Index

A

About
 function
 animate 37
 animatePro 37
 applicationPath 37
 controlCenterApp 37
 demoVersion 37
 educVersion 38
 fullVersion 38
 getApplicationPath 38
 getFlavorString 38
 getVersionInfoStr 38
 harmony 39
 interactiveApp 39
 isanimate 39
 isanimatePro 39
 isControlCenterApp 39
 isDemoVersion 40
 isEducVersion 40
 isFullVersion 40
 isHarmony 40
 isInteractiveApp 40
 isLinuxArch 41
 isMacArch 41
 isMacIntelArch 41
 isMacPpcArch 41
 isMainApp 41
 isPaintMode 42
 isScanApp 42
 isStage 42
 isWindowsArch 42
 isXsheetMode 42
 linuxArch 43
 macArch 43
 macIntelArch 43
 macPpcArch 43
 mainApp 43
 paintMode 44
 productName 44
 scanApp 44
 stage 44
 windowsArch 44
 xsheetMode 45

about
 scripting 9
 scripting templates 9
 accessing built-in objects
 Qt Script 17

Action
 function
 perform 46

C

creating
 Qt scripts 10, 12

E

export
 scripts 10, 14
 Exporting and Importing Scripts 10, 14

F

Function
 About 35
 summary of 18
 about
 animate 37
 animatePro 37
 isanimate 39
 isanimatePro 39
 about.applicationPath 37
 about.controlCenterApp 37
 about.demoVersion 37
 about.educVersion 38
 about.fullVersion 38
 about.getFlavorString 38
 about.getVersionInfoStr 38
 about.harmony 39
 about.interactiveApp 39
 about.isControlCenterApp 39
 about.isDemoVersion 40
 about.isEducVersion 40
 about.isFullVersion 40
 about.isHarmony 40
 about.isInteractiveApp 40
 about.isLinuxArch 41
 about.isMacArch 41
 about.isMacIntelArch 41
 about.isMacPpcArch 41
 about.isMainApp 41
 about.isPaintMode 42
 about.isScanApp 42
 about.isStage 42
 about.isWindowsArch 42
 about.isXsheetMode 42
 about.linuxArch 43
 about.macArch 43
 about.macIntelArch 43
 about.macPpcArch 43
 about.mainApp 43
 about.paintMode 44
 about.productName 44
 about.scanApp 44
 about.stage 44
 about.windowsArch 44
 about.xsheetMode 45
 Action
 summary of 20
 action.perform 46
 Column 47
 summary of 20
 column.add 48
 column.clearKeyFrame 48
 column.getColorForXSheet 48, 49
 column.getDisplayName 49
 column.getDrawingName 49
 column.getDrawingTimings 49
 column.getElementOfDrawing 50
 column.getEntry 50
 column.getName 50
 column.getNextKeyDrawing 50
 column.getTextOfExpr 51
 column.importSound 51
 column.isKeyFrame 51
 column.numberOf 51
 column.rename 52
 column.setColorForXSheet 52
 column.setElementOfDrawing 52

- column.setEntry [53](#)
- column.setKeyFrame [53](#)
- column.setTextOfExpr [53](#)
- column.type [54](#)
- CopyPaste
 - summary of [21](#)
- copyPaste.createTemplateFromSelection [55](#)
- copyPaste.pasteTemplateIntoScene [55](#)
- copyPaste.setPasteSpecialAddRemoveAngleKeyFrame [57](#)
- copyPaste.setPasteSpecialAddRemoveMotionKeyFrame [57](#)
- copyPaste.setPasteSpecialAddRemoveScalingKeyFrame [58](#)
- copyPaste.setPasteSpecialAddRemoveSkewKeyFrame [57](#)
- copyPaste.setPasteSpecialAddRemoveVelocityKeyFrame [57](#)
- copyPaste.setPasteSpecialColorPaletteOption [59](#)
- copyPaste.setPasteSpecialCreateNewColumn [56](#)
- copyPaste.setPasteSpecialDrawingAction [58](#)
- copyPaste.setPasteSpecialDrawingAutomaticExtendExposure [59](#)
- copyPaste.setPasteSpecialDrawingFileMode [59](#)
- copyPaste.setPasteSpecialElementTimingColumnMode [57](#)
- copyPaste.setPasteSpecialForcesKeyFrameAtBegAndEnd [58](#)
- copyPaste.setPasteSpecialOffsetKeyFrames [58](#)
- copyPaste.setPasteSpecialReplaceExpressionColumns [58](#)
- copyPaste.usePasteSpecial [56](#)
- Element [60](#)
 - summary of [22](#)
- element.add [60](#)
- element.element.pixmapFormat [61](#)
- element.element.remove [62](#)
- element.fieldChart [60](#)
- element.folder [61](#)
- element.getName [61](#)
- element.id [61](#)
- element.numberOf [49, 61](#)
- element.rename [62](#)
- element.scanType [62](#)
- element.vectorType [62](#)
- Exporter
 - summary of [22](#)
- exporter.cleanExportDir [63](#)
- exporter.getExportDir [63](#)
- Frame [64](#)
 - summary of [23](#)
- frame.current [64, 65](#)
- frame.insert [65](#)
- frame.numberOf [65](#)
- frame.remove [65](#)
- func.addCtrlPointAfter3DPath [68](#)
- func.addKeyFrame3DPath [68](#)
- func.angleEaseIn [68](#)
- func.angleEaseOut [69](#)
- func.holdStartFrame [69](#)
- func.holdStep [69](#)
- func.holdStopFrame [69](#)
- func.numberOfPoints [70](#)
- func.numberOfPoints3DPath [70](#)
- func.pointBias3DPath [70](#)
- func.pointConstSeg [70](#)
- func.pointContinuity [71](#)
- func.pointContinuity3DPath [71](#)
- func.pointEaseIn [71](#)
- func.pointEaseOut [71](#)
- func.pointHandleLeftX [72](#)
- func.pointHandleLeftY [72](#)
- func.pointHandleRightX [72](#)
- func.pointHandleRightY [72](#)
- func.pointLockedAtFrame [73](#)
- func.pointTension3DPath [73](#)
- func.pointX [73](#)
- func.pointX3DPath [73](#)
- func.pointY [74](#)
- func.pointY3DPath [74](#)
- func.pointZ3DPath [74](#)
- func.removePoint3DPath [74](#)
- func.setBezierPoint [75](#)
- func.setEasePoint [75](#)
- func.setHoldStartFrame [76](#)
- func.setHoldStep [76](#)
- func.setHoldStopFrame [76](#)
- func.setPoint3DPath [76](#)
- func.setVeloBasedPoint [77](#)
- Function Curve [66](#)
 - summary of [23](#)
- getApplicationPath [38](#)
- MessageLog
 - summary of [25](#)
- MessageLog.debug [78](#)
- MessageLog.isDebugEnabled [78](#)
- MessageLog.setDebug [78](#)
- MessageLog.trace [78](#)
- Node [79](#)
 - summary of [25](#)
- node. [81, 88, 89, 90](#)
- node.add [81](#)
- node.coordX [81](#)
- node.coordY [81](#)
- node.createGroup [82](#)
- node.deleteNode [82](#)
- node.dstNode [82](#)
- node.equals [82](#)
- node.explodeGroup [83](#)
- node.flatDstNode [83](#)
- node.flatSrcNode [83](#)
- node.getCameras [83](#)
- node.getDefaultCamera [84](#)
- node.getEnable [84](#)
- node.getMatrix [84](#)
- node.getName [84](#)
- node.getTextAttr [84](#)
- node.inkAttr [85](#)
- node.isGroup [85](#)
- node.link [85](#)
- node.linkedColumn [86](#)
- node.noNode [86](#)
- node.numberOfInputPorts [86](#)
- node.numberOfOutputLinks [86](#)
- node.numberOfOutputPorts [87](#)
- node.numberOfSubNodes [87](#)
- node.parentNode [87](#)
- node.rename [87](#)
- node.root [88](#)
- node.setAsDefaultCamera [88](#)
- node.setCoord [88](#)
- node.setEnable [88](#)
- node.setTextAttr [89](#)
- node.sLinked [85](#)
- node.srcNode [89](#)
- node.subNode [89](#)
- node.type [90](#)
- node.unlink [90](#)
- node.unlinkAttr [90](#)
- node.width/height [91](#)
- PaletteManager
 - summary of [27](#)

- PaletteManager.getColorId [94](#)
- PaletteManager.getColorName [94](#)
- PaletteManager.getCurrentColorId Function [92](#)
- PaletteManager.getCurrentColorName [92](#)
- PaletteManager.getCurrentPaletteld [92](#)
- PaletteManager.getCurrentPaletteName [93](#)
- PaletteManager.getCurrentPaletteSize [93](#)
- PaletteManager.getNumPalettes [94](#)
- PaletteManager.getPaletteld [95](#)
- PaletteManager.getPaletteName [94, 95](#)
- PaletteManager.setCurrentColorByld [93](#)
- PaletteManager.setCurrentPaletteAndColorByld [93](#)
- PaletteManager.setCurrentPaletteByld [93](#)
- PenStyleManager
 - summary of [28](#)
- PenstyleManager.changeCurrentPenstyleCenterlineSmoothness [98](#)
- PenstyleManager.changeCurrentPenstyleEraserFlag [98](#)
- PenstyleManager.changeCurrentPenstyleMaximumSize [98](#)
- PenstyleManager.changeCurrentPenstyleMinimumSize [98](#)
- PenstyleManager.changeCurrentPenstyleOutlineSmoothness [98](#)
- PenstyleManager.exportPenstyleListToString [100](#)
- PenstyleManager.exportPenstyleToString [100](#)
- PenstyleManager.getCurrentPenstyleCenterlineSmoothness [99](#)
- PenstyleManager.getCurrentPenstyleEraserFlag [100](#)
- PenstyleManager.getCurrentPenstyleIndex [99](#)
- PenstyleManager.getCurrentPenstyleMaximumSize [99](#)
- PenstyleManager.getCurrentPenstyleMinimumSize [99](#)
- PenstyleManager.getCurrentPenstyleName [97](#)
- PenstyleManager.getCurrentPenstyleOutlineSmoothness [99](#)
- PenstyleManager.getNumberOfPenstyles [97](#)
- PenstyleManager.getPenstyleName [97](#)
- PenstyleManager.importPenstyleListFromString [100](#)
- PenstyleManager.savePenstyles [100](#)
- PenstyleManager.setCurrentPenstyleByIndex [97](#)
- PenstyleManager.setCurrentPenstyleByName [97](#)
- Preferences [101](#)
 - summary of [29](#)
- preferences.getBool [101](#)
- preferences.getColor [102](#)
- preferences.getDouble [102](#)
- preferences.getInt [102](#)
- preferences.getString [102](#)
- preferences.setBool [102](#)
- preferences.setColor [103](#)
- preferences.setDouble [103](#)
- preferences.setInt [103](#)
- preferences.setString [103](#)
- Render
 - summary of [29](#)
- render.cancelRender [107](#)
- render.frameReady [105](#)
- render.renderFinished [105](#)
- render.renderScene [106](#)
- render.renderSceneAll [107](#)
- render.setBgColor [106](#)
- render.setCombine [105](#)
- render.setFieldType [105](#)
- render.setRenderDisplay [106](#)
- render.setResolution [106](#)
- render.setWriteEnabled [106](#)
- Scene [108](#)
 - summary of [30](#)
- scene. [112](#)
- scene.beginUndoRedoAccum [109](#)
- scene.cancelUndoRedoAccum [109](#)
- scene.clearHistory [109](#)
- scene.coordAtCenterX [110](#)
- scene.coordAtCenterY [110](#)
- scene.currentEnvironment [110](#)
- scene.currentJob [110](#)
- scene.currentProjectPath [110](#)
- scene.currentProjectPathRemapped [111](#)
- scene.currentResolutionX [111](#)
- scene.currentResolutionY [111](#)
- scene.currentScene [111](#)
- scene.currentVersion [111](#)
- scene.defaultResolutionFOV [112](#)
- scene.defaultResolutionX [112](#)
- scene.defaultResolutionY [112](#)
- scene.endUndoRedoAccum [112](#)
- scene.fromOGL [113](#)
- scene.getCameraMatrix [113](#)
- scene.getFrameRate [113](#)
- scene.numberOfUnitsX [113](#)
- scene.numberOfUnitsY [113](#)
- scene.numberOfUnitsZ [114](#)
- scene.saveAll [114](#)
- scene.saveAsNewVersion [114](#)
- scene.setCoordAtCenter [114](#)
- scene.setDefaultResolution [115](#)
- scene.setFrameRate [116](#)
- scene.setNumberofUnits [114](#)
- scene.setUnitsAspectRatio [115](#)
- scene.toOGL [115](#)
- scene.unitsAspectRatioX [115](#)
- scene.unitsAspectRatioY [115](#)
- Selection [117](#)
 - summary of [31](#)
- selection.addColumnToSelection [118](#)
- selection.addDrawingColumnToSelection [118](#)
- selection.addNodeToSelection [118](#)
- selection.clearSelection [118](#)
- selection.extendSelectionWithColumn [118](#)
- selection.numberofColumnsSelected [119](#)
- selection.numberofFramesSelected. [119](#)
- selection.numberofnodesslected [119](#)
- selection.selectAll [119](#)
- selection.selectedColumn [119](#)
- selection.selectedNode [120](#)
- selection.setSelectionFrameRange [120](#)
- Sound
 - summary of [32](#)
- sound.getSoundtrack [122](#)
- sound.getSoundtrackAll [122](#)
- sound.setChannelCount [121](#)
- sound.setChannelSize [121](#)
- sound.setSampleRate [121](#)
- SpecialFolders [123](#)
 - summary of [32](#)
- specialFolders.app [123](#)
- specialFolders.bin [124](#)
- specialFolders.config [124](#)
- specialFolders.etc [124](#)
- specialFolders.foot [126](#)
- specialFolders.htmlHelp [124](#)
- specialFolders.lang [124](#)
- specialFolders.library [125](#)
- specialFolders.pdf [125](#)
- specialFolders.platform [125](#)
- specialFolders.plugins [125](#)
- specialFolders.resource [125](#)
- specialFolders.temp [126](#)

specialFolders.userConfig [126](#)
 Timeline [127](#)
 summary of [33](#)
 Timeline. [131](#)
 Timeline.firstFrameSel [129](#)
 Timeline.isAncestorOf [129](#)
 Timeline.layerIsColumn [129](#)
 Timeline.layerIsNode [129](#)
 Timeline.layerToColumn [130](#)
 Timeline.layerToNode [130](#)
 Timeline.numFrameSel [130](#)
 Timeline.numLayers [130](#)
 Timeline.numLayerSel [130](#)
 Timeline.parentNodeIndex [131](#)
 Timeline.selIsColumn [131](#)
 Timeline.selToColumn [131](#)
 Timeline.selToLayer [132](#)
 Timeline.selToNode [132](#)
 Timeline.setDisplayToUnconnected [132](#)
 View [133](#)
 summary of [34](#)
 view.column [133](#)
 view.currentView [133](#)
 view.group [133](#)
 view.refreshViews [134](#)
 view.type [134](#)

Function Summary [18](#)

I

import
 scripts [10, 14](#)

L

Linking a Script to a Toolbar Button [11, 15](#)

Q

QT script
 using [9](#)

Qt Script

 accessing built-in objects using [17](#)
 access to About Function [18, 35](#)
 access to Action Function [20](#)
 access to Column Function [20, 47](#)
 access to CopyPaste Function [21](#)
 access to Element Function [22, 60](#)
 access to Exporter Function [22](#)
 access to Frame Function [23, 64](#)
 access to Function Curve Function [23, 66](#)
 access to MessageLog Function [25](#)
 access to Node Function [25, 79](#)
 access to PaletteManager Function [27](#)
 access to PenStyleManager Function [28](#)
 access to Preferences Function [29, 101](#)
 access to Render Function [29](#)
 access to Scene Function [30, 108](#)
 access to Selection Function [31, 117](#)
 access to Sound Function [32](#)
 access to SpecialFolders Function [32, 123](#)
 access to Timeline Function [33, 127](#)
 access to View Function [34, 133](#)

Qt scripts
 creating [10, 12](#)

S

script
 link to toolbar button [11, 15](#)
 scripting

about [9](#)
 list of functions [15, 17](#)
 list of objects [15, 17](#)
 template
 about [9](#)
 using QT script [9](#)
 Sound
 function
 getSoundtrack [122](#)
 getSoundtrackAll [122](#)
 setChannelCount [121](#)
 setChannelSize [121](#)
 setSampleRate [121](#)
 functions [121](#)

T

templates
 scripting [9](#)

