
# ANIMATE


# ANIMATE


## 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



## Trademarks



## 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

## Publication Date

April 2010

# Contents

# 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 and Toon Boom Animate Pro.

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



## Scripting Overview

The following topics are addressed:



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

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

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 the software, you should do the following:

**For Animate:**

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.

**For Animate Pro:**

1. Download the sample Scripting template available in the Documentation section of the Toon Boom website at:

**http://www.toonboom.com/products/animatepro/eLearning/**

2. Once you have downloaded the package, uncompress it and read the instructions contained in the package.

### Using Qt Script to Automate the Functions

Qt Script provides access to many of the functionalities supported in the 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.

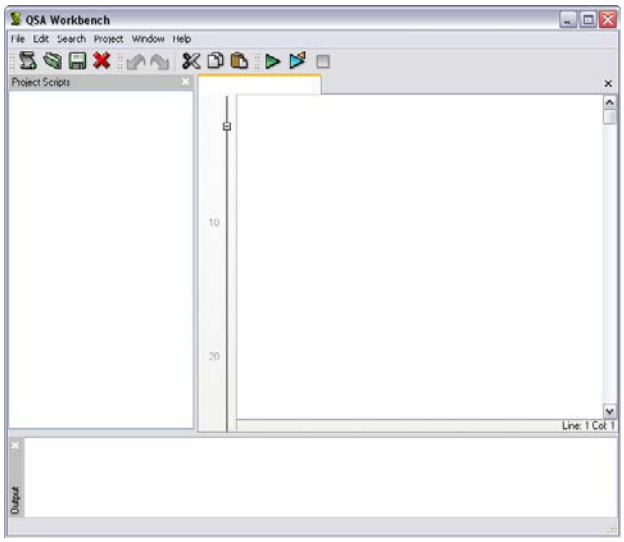
# Creating a Qt Script

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

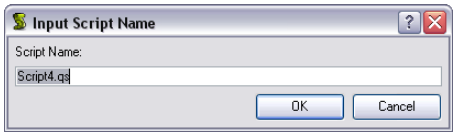
**To create a script:**

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.



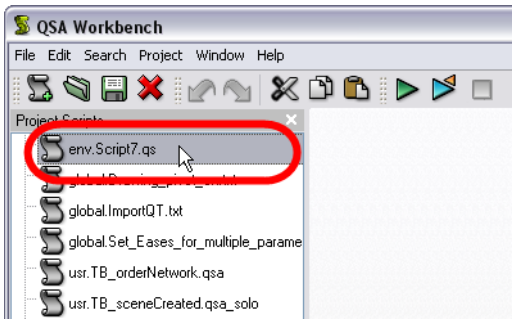

3. Enter a name in the **Script Name** field and click on **OK**.

# 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**.

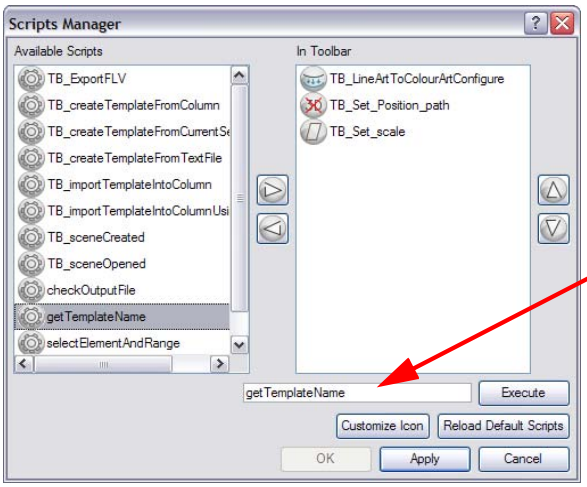
   The script is imported and appears in the Project Scripts list. It is automatically saved at the user level (all scripts are `usr.Scriptname.qs`).

## 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.

# Server Network Environment

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



## Using Qt Script to Automate Functions - server functions

Qt Script provides access to many of the functionalities supported in the software interface 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 the software.

- **Animate**:
  By default, all scripts are saved in:

  ⇨ **On Mac OS X**

  `/Users/Current_User_Name/Library/Preferences/Toon Boom Animation/Toon Boom Animate/790-scripts/Script.qsa`

  ⇨ **On Windows 7/ Vista**

  `C:\Users\Current_User_Name\AppData\Roaming\Toon Boom Animation\Toon Boom Animate\790-scripts`

  ⇨ **On Windows XP**

  `C:\Documents and Settings\Current_User_Name\Application Data\Toon Boom Animation\Toon Boom Animate\790-scripts`

- **Animate Pro**:
  By default, all scripts are saved in:

  ⇨ **On Mac OS X**

  `/Users/Current_User_Name/Library/Preferences/Toon Boom Animation/Toon Boom Animate Pro/790-scripts/Script.qsa`

  ⇨ **On Windows 7/ Vista**

  `C:\Users\Current_User_Name\AppData\Roaming\Toon Boom Animation\Toon Boom Animate Pro\790-scripts`

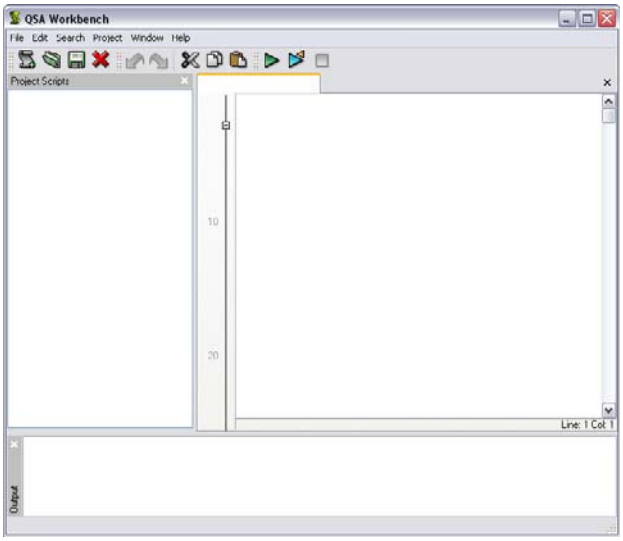
  ⇨ **On Windows XP**

  `C:\Documents and Settings\Current_User_Name\Application Data\Toon Boom Animation\Toon Boom Animate Pro\790-scripts`

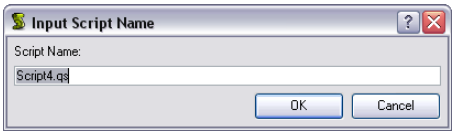
**To create a script:**

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

The QSA Workbench dialog box opens.

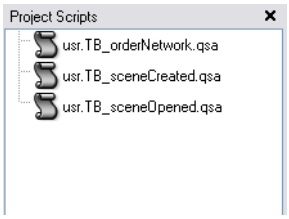



2. Select **File > New Script**  . The Input Script Name dialog box appears.



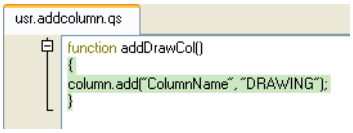

3. Enter a name in the **Script Name** field and click on **OK**.

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

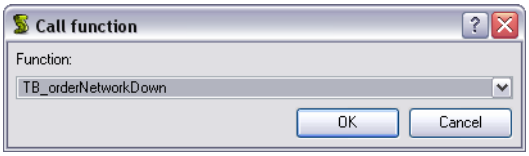



4. Double-click on the script name in the Project Scripts list panel. A tab with a text box appears to the right of the Projects Scripts panel.
5. Write your script in the tabbed text box. Try the following script:

```
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.

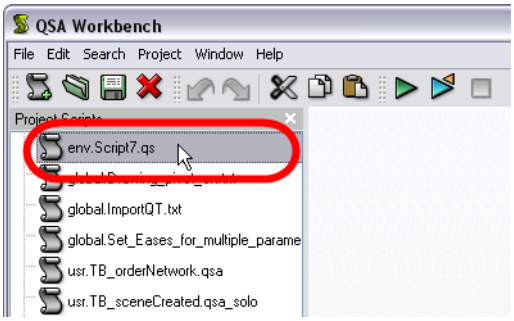
- If a syntax error occurs, it will be displayed in the Output text box.

## 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**.

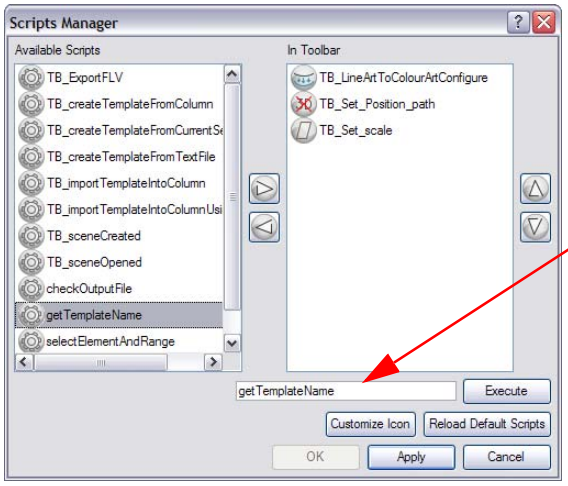
   The script is imported and appears in the Project Scripts list.

## 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

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

# Chapter 2
# Scripting Reference

The Scripting Reference chapter applies to both the Toon Boom Animate and Toon Boom Animate Pro. This chapter describes the software's built-in objects which can be accessed with Qt® Script.

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



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/qsa-1.1.5/language.html**
- Creating Qt Scripts, including building dialog boxes:
  **http://doc.trolltech.com/qsa-1.1.5/qsa-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 | Description |
|---|---|
| **about.animate** | This is a read-only property that is true whenever this script is being executed by Animate For details, see animate, on page 37. |
| **about.animatePro** | 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. |
| **about.applicationPath;** | Contains application path. For details, see applicationPath, on page 37. |
| **about.controlCenterApp;** | Contains true when ControlCenter. For details, see controlCenterApp, on page 37. |
| **about.demoVersion;** | Contains true when a Demo Version. For details, see demoVersion, on page 37. |
| **about.educVersion;** | Contains true when in Educational Version. For details, see educVersion, on page 38. |
| **about.fullVersion;** | Contains true when in Commercial Version. For details, see fullVersion, on page 38. |
| **about.getApplicationPath();** | Returns the application path. For details, see getApplicationPath, on page 38. |
| **about.getFlavorString();** | Returns the flavour string. For details, see getFlavorString, on page 38. |
| **about.getVersionInfoStr();** | Returns the version string. For details, see getVersionInfoStr, on page 38. |
| **about.harmony;** | Returns true when in Harmony. For details, see harmony, on page 39. |
| **about.interactiveApp** | Contains true when interactive application. For details, seeinteractiveApp, on page 39. |
| **about.isAnimate()** | Returns true when in Animate. For details, see isAnimate, on page 39. |

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

| About Function | Description |
|---|---|
| `about.windowsArch;` | Contains true when on Windows. For details, see windowsArch, on page 44. |
| `about.xsheetMode;` | Contains true when in XSheet Mode. For details, see xsheetMode, on page 45. |

| Action Function | Description |
|---|---|
| `action.perform(name);` | This function calls the menu function directly. For details, see perform, on page 46. |

| Column Function | Description |
|---|---|
| `column.add("columnName", "columnType");` | Adds a column with the specified name and type. For details, see add, on page 48. |
| `column.clearKeyFrame("columnName", atFrame );` | Removes a keyframe from a cell in a column. For details, see clearKeyFrame, on page 48. |
| `column.getColorForXSheet();` | Returns the colour of a specified XSheet column. For details, see getColorForXSheet, on page 48. |
| `column.getColumnListOfType("DRAWING");` | Returns the name of all drawing columns For details, see getColumnListOfType, on page 48. |
| `column.getCurrentVersionForDrawing();` | Returns the current drawing version for the given column and drawing name. For details, see getCurrentVersionFor Drawing, on page 49. |
| `column.getDisplayName();` | 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 getDisplayName, on page 49. |
| `column.getDrawingName();` | Returns the name at given frame of a drawing column. For details, see getDrawingName, on page 49. |
| `column.getDrawingTimings("columnName", true);` | Lists list of all drawing names used in the drawing column For details, see getDrawingTimings, on page 49. |
| `column.getElementIdOfDrawing("columnName");` | Returns the id of the element linked to the specified drawing column. For details, see getElementIdOfDrawing, on page 50. |
| `column.getEntry("columnName", subColumn, atFrame);` | Returns the value of a cell in a column. For details, see getEntry, on page 50. |
| `column.getName(columnNumber);` | Returns the unique identifier that names the column. This is not the display name shown in the xsheet view. For details, see getName, on page 50. |
| `column.getTextOfExpr("columnName");` | Returns the expression text in the identified column. For details, see getTextOfExpr, on page 51. |
| `column.importSound();` | Import a external sound file into a sound column. For details, see importSound, on page 51. |
| `column.isKeyFrame("columnName", subColumn, atFrame);` | Returns true or false indicating if a cell in a column is a keyframe. For details, see isKeyFrame, on page 51. |
| `column.getNextKeyDrawing();` | Returns the frame number of the next key drawing in the given drawing column from the specified start frame.For details, see getNextKeyDrawing, on page 50. |
| `column.numberOf();` | Returns the number of columns in the scene. For details, see numberOf, on page 51. |

| Column Function | Description |
|---|---|
| `column.rename("oldName", "newName");` | Renames the specified column. For details, see rename, on page 52. |
| `column.setColorForXSheet();` | Change the colour of an XSheet column. For details, see setColorForXSheet, on page 52. |
| `column.setElementIdOfDrawing("columnName", "elementId");` | Links an empty Drawing column to an element. For details, see setElementIdOfDrawing, on page 52. |
| `column.setEntry("columnName", subColumn, atFrame, "value" );` | Set the value of a cell in a column. For details, see setEntry, on page 53. |
| `column.setKeyFrame("columnName", atFrame );` | Makes a cell in a column a keyframe. For details, see setKeyFrame, on page 53. |
| `column.setTextOfExpr("columnName", "text");` | Sets the value in the Expression column to the specified text. For details, see setTextOfExpr, on page 53. |
| `column.type("columnName");` | 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 |
|---|---|
| `copyPaste.createTemplateFromSelection( "name", const "path" );` | This function creates template from the current selection in the scene, using only the current drawing versions. For details, see createTemplateFromSelection, on page 55. |
| `copyPaste.pasteTemplateIntoScene( const "templateSrcPath", "insertColumnName", int insertFrame );` | This function pastes the template into the scene. For details, see pasteTemplateIntoScene, on page 55. |
| `copyPaste.usePasteSpecial( flag );` | This function enables the paste special options. For details, see pasteTemplateIntoScene, on page 55. |
| `copyPaste.setExtendScene( flag );` | This function sets the option to extend the scene to accommodate the incoming template. For details, see pasteTemplateIntoScene, on page 55. |
| `copyPaste.setPasteSpecialCreateNewColumn( flag );` | This function sets the option to create new columns to paste. For details, see setPasteSpecialCreateNewColumn, on page 56. |
| `copyPaste.setPasteSpecialElementTimingColumnMode ( "mode" );` | This function sets the paste special element timing mode for calls to pasteTemplateIntoScene. For details, see setPasteSpecialElementTimingColumnMode, on page 57. |
| `copyPaste.setPasteSpecialAddRemoveMotionKeyFrame( flag );` | PEGS: This function is a paste special option for pegs and functions. For details, see setPasteSpecialAddRemoveMotionKeyFrame, on page 57. |
| `copyPaste.setPasteSpecialAddRemoveVelocityKeyFrame( flag );` | PEGS: This function is a paste special option for pegs and functions.For details, see setPasteSpecialAddRemoveVelocityKeyFrame, on page 57. |
| `copyPaste.setPasteSpecialAddRemoveAngleKeyFrame( flag );` | PEGS: This function is a paste special option for pegs and functions.For details, see setPasteSpecialAddRemoveAngleKeyFrame, on page 57. |
| `copyPaste.setPasteSpecialAddRemoveSkewKeyFrame( flag );` | PEGS: This function is a paste special option for pegs and functions.For details, see setPasteSpecialAddRemoveScalingKeyFrame, on page 58. |
| `copyPaste.setPasteSpecialAddRemoveScalingKeyFrame( flag );` | PEGS: This function is a paste special option for pegs and functions.For details, see setPasteSpecialAddRemoveScalingKeyFrame, on page 58. |

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

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

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

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

| Function Curve Function | Description |
|---|---|
| `func.holdStartFrame("columnName");` | 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. |
| `func.holdStep("columnName");` | 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. |
| `func.holdStopFrame("columnName");` | 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** | |
| `func.numberOfPoints("columnName");` | Returns the number of keyframes and control points on a curve. For details, see numberOfPoints, on page 70. |
| `func.pointConstSeg("columnName", point);` | 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. |
| `func.pointContinuity("columnName", point);` | 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. |
| `func.pointX("columnName", point);` | Returns the X value (frame number) of a point on a function curve. For details, see pointX, on page 73. |
| `func.pointY("columnName", point);` | Returns the Y value of a point on a function curve. For details, see pointY, on page 74. |
| **Bezier Functions** | |
| `func.pointHandleLeftX("columnName", point);` | Returns the X value of the left handle of a point on a curve. For details, see pointHandleLeftX, on page 72. |
| `func.pointHandleLeftY("columnName", point);` | Returns the Y value of the left handle of a point on a curve. For details, see pointHandleLeftY, on page 72. |
| `func.pointHandleRightX("columnName", point);` | Returns the X value of the right handle of a point on a curve. For details, see pointHandleRightX, on page 72. |
| `func.pointHandleRightY("columnName", point);` | Returns the Y value of the right handle of a point on a curve. For details, see pointHandleRightY, on page 72. |
| **Ease Functions** | |
| `func.angleEaseIn("columnName", point);` | Returns the angle of the ease-in handle. For details, see angleEaseIn, on page 68. |

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

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

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

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

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

| Node Function | Description |
|---|---|
| `node.setTextAttr("node_path", "attrName", atFrame, "attrValue");` | Changes the value of an attribute in a module. For details, see setTextAttr, on page 89. |
| `node.srcNode("node_path", iPort);` | Returns the path for the module that the port is linked to. For details, see srcNode, on page 89. |
| `node.subNode("node_path_parent", iSubNode);` | Returns the path of a module in a group. For details, see subNode, on page 89. |
| `node.subNodeByName ();` | Returns the path of a module in a group. For details, see subNodeByName, on page 90. |
| `node.type("node_path");` | Returns the module type. For details, see type, on page 90. |
| `node.unlink("dstNode_path", inPort);` | Unlinks a port on one module from the port on another module. For details, see unlink, on page 90. |
| `node.unlinkAttr("node_path", "attrName");` | Unlinks an attribute from a function column. For details, see unlinkAttr, on page 90. |
| `node.height();` | Returns the node height. For details, see width/height, on page 91. |
| `node.width();` | Returns the node width. For details, see width/height, on page 91. |

| PaletteManager Function | Description |
|---|---|
| `PaletteManager.getCurrentColorId();` | This function retrieves the id of the currently selected colour.. For details, see getCurrentColorId, on page 92. |
| `PaletteManager.getCurrentColorName();` | This function returns the current colour name from the ColourView. For details, see getCurrentColorName, on page 92. |
| `PaletteManager.getCurrentPaletteId();` | This function returns the id of the current palette from the ColourView. For details, see getCurrentPaletteId, on page 92. |
| `PaletteManager.getCurrentPaletteName();` | This function returns the current palette name from the ColourView. For details, see getCurrentPaletteName, on page 93. |
| `PaletteManager.setCurrentPaletteById( "palette");` | This function sets the current palette in the ColourView. For details, see setCurrentPaletteById, on page 93. |
| `PaletteManager.setCurrentColorById( "color");` | This function sets the current colour in the ColourView. For details, see setCurrentColorById, on page 93. |
| `PaletteManager.setCurrentPaletteAndColorById("palette", "color");` | This function sets the current palette and colour in the ColourView. For details, see setCurrentPaletteAndColorById, on page 93. |
| `PaletteManager.getCurrentPaletteSize();` | This function returns the size of the currently selected palette in the ColourView. For details, see getCurrentPaletteSize, on page 93. |
| `PaletteManager.getColorName(int index);` | This function returns the name of the colour in the currently selected palette. For details, see getColorName, on page 94. |
| `PaletteManager.getColorId(int index);` | This function returns the id of the colour in the currently selected palette. For details, see getColorId, on page 94. |
| `PaletteManager.getNumPalettes();` | This function returns the number of palettes in the current selected palette list in ColourView list. For details, see getNumPalettes, on page 94. |
| `PaletteManager.getNumPalettes( scenePaletteList );` | This function returns the number of palettes in palette list in ColourView. For details, see getNumPalettes, on page 94. |

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

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

| PenStyleManager Function | Description |
|---|---|
| `PenstyleManager.exportPenstyleList ToString();` | 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 exportPenstyleListToString, on page 100. |
| `PenstyleManager.importPenstyleList FromString("str");` | This function imports a penstyle list from a previously formatted penstyle string. For details, see importPenstyleListFromString, on page 100. |
| `PenstyleManager.savePenstyles();` | This function saves the pen styles. For details, see getNumberOfPenstyles, on page 97. |
| `PenstyleManager.savePenstyles();` | This function saves the pen styles. For details, see getNumberOfPenstyles, on page 97. |

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

| Render Function | Description |
|---|---|
| `render.frameReady(int frame, SM_CelWrapper &frameCel);` | Event that notifies the script that a certain frame is available and at which location. For details, see frameReady, on page 105. |
| `render.renderFinished();` | Event that notifies the script when the render has completed. For details, see renderFinished, on page 105. |
| `render.setCombine(autoCombine, secondFieldFirst);` | 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 setCombine, on page 105. |
| `render.setFieldType(int type);` | Sets the frame output format. For details, see setFieldType, on page 105. |
| `render.setBgColor(QColor bgColor);` | Set the background colour to use when rendering in scene machine mode. For details, see setBgColor, on page 106. |
| `render.setResolution(int x, int y);` | Set the scene resolution to use for rendering. For details, see setResolution, on page 106. |
| `render.setRenderDisplay("name");` | Set which display module to use for rendering."Display All" uses the global unconnected display module. For details, see setRenderDisplay, on page 106. |

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

| Scene Function | Description |
|---|---|
| `scene.beginUndoRedoAccum("commandname");` | 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. For details, see beginUndoRedoAccum, on page 109. |
| `scene.cancelUndoRedoAccum();` | Cancel the accumulation of undo commands. Rollback all executed commands. For details, see cancelUndoRedoAccum, on page 109. |
| `scene.clearHistory` | This function clears the command history. After this call, it is not possible to undo the command.For details, see clearHistory, on page 109. |
| `scene.coordAtCenterX();` | Returns the X value of the centre coordinate of the scene grid. For details, see coordAtCenterX, on page 110. |
| `scene.coordAtCenterY();` | Returns the Y value of centre coordinate of the scene grid. For details, see coordAtCenterY, on page 110. |
| `scene.currentEnvironment();` | Returns the current environment name.For details, seecurrentEnvironment, on page 110. |
| `scene.currentJob();` | Returns the current job name. For details, see currentJob, on page 110. |
| `scene.currentProjectPath();` | Returns the current project path. For details, seecurrentProjectPath, on page 110. |
| `scene.currentProjectPathRemapped();` | Returns the current project path with shortcut translated and symlink resolved. For details, see currentProjectPathRemapped, on page 111. |
| `scene.currentResolutionX();` | Returns the preview resolution in X. For details, see currentResolutionX, on page 111. |
| `scene.currentResolutionY();` | Returns the previous resolution in Y. For details, see currentResolutionY, on page 111. |
| `scene.currentScene();` | Returns the name of the current scene. For details, see currentScene, on page 111. |
| `scene.currentVersion();` | Returns the current version of project/scene. For details, see currentVersion, on page 111. |
| `scene.defaultResolutionFOV();` | Returns the field of view. For details, seedefaultResolutionFOV, on page 112. |
| `scene.defaultResolutionName();` | Returns the name of the scene resolution preset. For details, see defaultResolutionName, on page 112. |
| `scene.defaultResolutionX();` | Returns the scene resolution in X. For details, see defaultResolutionX, on page 112. |
| `scene.defaultResolutionY();` | Returns the scene resolution in Y. For details, see defaultResolutionY, on page 112. |

| Scene Function | Description |
|---|---|
| `scene.endUndoRedoAccum();` | 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. For details, see endUndoRedoAccum, on page 112. |
| `scene.fromOGL();` | Convert an OGL coordinate to a field coordinate. For details, see fromOGL, on page 113. |
| `scene.getCameraMatrix();` | Returns the model matrix for the default camera. Note that this is the inverse of the view matrix. For details, see getCameraMatrix, on page 113. |
| `scene.getFrameRate();` | Returns the frame rate. For details, see getFrameRate, on page 113. |
| `scene.numberOfUnitsX();` | Returns the number of units in the X-axis of the scene grid. For details, see numberOfUnitsX, on page 113. |
| `scene.numberOfUnitsY();` | Returns the number of units in the Y-axis of the scene grid. For details, see numberOfUnitsY, on page 113. |
| `scene.numberOfUnitsZ();` | Returns the number of units in the Z-axis of the scene grid. For details, see numberOfUnitsZ, on page 114. |
| `scene.saveAll();` | Save project, all drawings, palettes and palette list. For details, see saveAll, on page 114. |
| `scene.saveAsNewVersion("name.true");` | Save all, while creating a new version of project. For details, see saveAsNewVersion, on page 114. |
| `scene.setUnitsAspectRatio(x, y);` | Sets the aspect ratio of the scene. For details, see setUnitsAspectRatio, on page 115. |
| `scene.setNumberOfUnits(x, y, z);` | Sets the number of X, Y, and Z units in the scene grid. For details, see setNumberofUnits, on page 114. |
| `scene.setCoordAtCenter(x, y);` | Sets the value of the centre (X, Y) coordinates. For details, see setCoordAtCenter, on page 114. |
| `scene.setDefaultResolution();` | Set the default resolution X,Y and field of view. For details, see setDefaultResolution, on page 115. |
| `scene.setFrameRate();` | Set the global frame rate. For details, seesetFrameRate, on page 116. |
| `scene.toOGL();` | Converts a field coordinate to an OGL coordinate. For details, see toOGL, on page 115. |
| `scene.unitsAspectRatioX();` | Returns the X value of the aspect ratio of the cells in the scene grid. For details, see unitsAspectRatioX, on page 115. |
| `scene.unitsAspectRatioY();` | Returns the Y value of the aspect ratio of the cells in the scene grid. For details, see unitsAspectRatioY, on page 115. |

| Selection Function | Description |
|---|---|
| `selection.clearSelection();` | This function clears the selection. For details, see clearSelection, on page 118. |
| `selection.addDrawingColumnToSelection(columnName);` | This function adds the drawing column and it's associated read node to the selection. For details, see addDrawingColumnToSelection, on page 118 |
| `selection.addColumnToSelection(column);` | This function adds a column to the selection.For details, see addColumnToSelection, on page 118. |
| `selection.addNodeToSelection(node);` | This function adds a node to the selection. For details, see addNodeToSelection, on page 118. |

| Selection Function | Description |
|---|---|
| `selection.extendSelectionWithColumn(columnName);` | This function adds the drawing column to the selection.<br>If the column is a drawing column, also adds the associated read node to the selection. For details, see extendSelectionWithColumn, on page 118. |
| `selection.numberOfCellColumnsSelected();` | This function returns a value for the number of selected columns. For details, see numberOfCellColumnsSelected, on page 119. |
| `selection.numberOfFramesSelected();` | This function returns the number of frames -selected (to be used in the xhseetview only). For details, see numberOfFramesSelected, on page 119. |
| `selection.numberOfNodesSelected();` | This function returns the number of modules that are selected. For details, see numberOfNodesSelected, on page 119. |
| `selection.selectAll();` | This function selects all nodes and all columns in the scene. For details, see selectAll, on page 119. |
| `selection.selectedCellColumn(int i);` | This function returns the name of the selected column. For details, see selectedCellColumn, on page 119. |
| `selection.selectedNode(int i);` | This function returns the path of the selected node. For details, see selectedNode, on page 120. |
| `selection.setSelectionFrameRange(int start, int end );` | This function sets the frame range for the selection. For details, see setSelectionFrameRange , on page 120. |

| Sound Function | Description |
|---|---|
| `sound.setSampleRate(double rate);` | This function sets the audio sample rate. For details, see setSampleRate, on page 121. |
| `sound.setChannelSize(int size);` | Sets the audio channel size (i.e. 8 or 16 bit). For details, see setChannelSize, on page 121. |
| `sound.setChannelCount(int count);` | Sets the number of audio channels (i.e 1 for mono and 2 for stereo). For details, see setChannelCount, on page 121. |
| `sound.getSoundtrack(int fromFrame, int toFrame);` | Returns a part of the scene's soundtrack in a temporary file in WAV format. For details, see getSoundtrack, on page 122. |
| `sound.getSoundtrackAll();` | Returns the scene's soundtrack in a temporary file in WAV format. For details, see getSoundtrackAll, on page 122. |

| SpecialFolders Function | Description |
|---|---|
| `specialFolders.app;` | Returns the global application folder. For details, see app, on page 123. |
| `specialFolders.bin;` | Returns the global binary folder. For details, see bin, on page 124. |
| `specialFolders.config;` | Returns the global configuration file folder. For details, see config, on page 124. |
| `specialFolders.etc;` | Returns the global./etc folder. For details, see etc, on page 124. |
| `specialFolders.htmlHelp;` | Returns the path to html folder. For details, see htmlHelp, on page 124. |
| `specialFolders.lang;` | Returns the language folder. For details, see lang, on page 124. |
| `specialFolders.library;` | Returns the platform library folder. For details, see library, on page 125. |
| `specialFolders.pdf` | Returns the path to pdf folder. For details, see platform, on page 125. |
| `specialFolders.platform;` | Returns the platform folder. For details, see platform, on page 125. |
| `specialFolders.plugins;` | Returns the platform plug-in folder. For details, see plugins, on page 125. |

| SpecialFolders Function | Description |
|---|---|
| `specialFolders.resource;` | Returns the resource folder. For details, see resource, on page 125. |
| `specialFolders.root;` | Returns the installation root folder. For details, see root, on page 126. |
| `specialFolders.temp;` | Returns the platform temporary folder. For details, see temp, on page 126. |
| `specialFolders.userConfig;` | Returns the user configuration file folder. For details, see userConfig, on page 126. |

| Timeline Function | Description |
|---|---|
| `Timeline.firstFrameSel` | 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. |
| `Timeline.isAncestorOf(parentLayerIdx, layerIdx);` | Returns true or false to identify if a layer is the parent of another layer. For details, see isAncestorOf, on page 129. |
| `Timeline.layerIsColumn(layerIdx);` | 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. |
| `Timeline.layerIsNode(layerIdx);` | 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. |
| `Timeline.layerToColumn(layerIdx);` | 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. |
| `Timeline.layerToNode(layerIdx);` | 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. |
| `Timeline.numFrameSel` | Returns the number of the selected frame in the Timeline, if only one frame is selected. For details, see numFrameSel, on page 130. |
| `Timeline.numLayers` | Returns the number of layers in the Timeline. For details, see numLayers, on page 130. |
| `Timeline.numLayersSel` | Returns the number of layers that are selected in the Timeline. For details, see numLayerSel, on page 130. |
| `Timeline.parentNodeIndex(layerIdx)` | Returns a layer identifier (layerIdx) for the parent of the layer (layerIdx). For details, see parentNodeIndex, on page 131. |
| `Timeline.selIsColumn(selIdx)` | Returns true or false to indicate if the Timeline selection has a column in the Xsheet. For details, see selIsColumn, on page 131. |
| `Timeline.selIsNode(selIdx)` | 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. |
| `Timeline.selToColumn(selIdx)` | Returns the column name for the Timeline selection. For details, see selToColumn, on page 131. |
| `Timeline.selToLayer(selIdx)` | Returns the layer identifier (layerIdx) for the selection (selIdx) in the Timeline. For details, see selToLayer, on page 132. |
| `Timeline.selToNode(selIdx)` | Returns the node name from the module in the Network for the Timeline selection. For details, see selToNode, on page 132. |
| `Timeline.setDisplayToUnconnected();` | Make the "Display All" display the current display. This method is obsolete. For details, see setDisplayToUnconnected, on page 132. |

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

**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:



**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 the Toon Boom software.

```
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, QUARTERNION 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

**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
- **`insertColumnName`**: - 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.
- **keyfrasmeMode**: 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:



**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:



**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:

- 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.

## setEnable

**Description**

This function sets the enable flag of the node.

**Syntax**

```
node.setEnable(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

**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

## getCurrentPaletteId

**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

## getPaletteId

**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

## getPaletteId

**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:



**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:



**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:

**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 * numberOfUnitsX()/Y * 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:



**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:

**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.

## selIsColumn

**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.

## selIsNode

**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:



**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



## C



## E



## F