macromedia®

FIREWORKS® 3

Extending Fireworks

macromedia™

Acknowledgments

Project Management: Margaret Dumas

Writing: Barbara Herbert

Production: Chris Basmajian

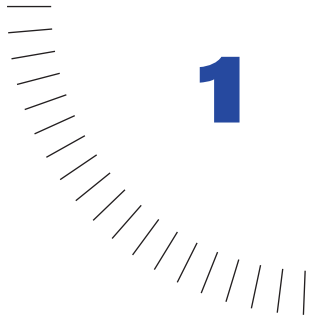
Special thanks to John Ahlqist, Dennis Griffin, Steve Johnson, Randy Varnell, Monte Williams

First Edition: December 1999

Macromedia, Inc.
600 Townsend St.
San Francisco, CA 94103

CONTENTS

CHAPTER 3



CHAPTER 1
Extending Fireworks Overview

To extend Fireworks, you must write JavaScript™ code. You can use JavaScript to write your own objects, behavior actions, and commands that affect Fireworks documents and the elements within them. To accomplish these tasks, you must be proficient in JavaScript and you must thoroughly understand Fireworks.

This manual describes the Document Object Model (DOM) that Fireworks supports and the Fireworks JavaScript API—the custom JavaScript functions that are built into Fireworks.

Prerequisites

Because Fireworks extensions must be written in JavaScript, this manual assumes that readers are familiar with JavaScript syntax and with basic programming concepts such as functions, arguments, and data types. It also assumes that readers understand the concept of working with objects and properties. This manual does not attempt to teach programming in general or JavaScript in particular.

Anyone who wants to extend Fireworks should have a good JavaScript reference to help with syntax questions (for example, is it substring() or subString()?). Useful JavaScript references include *JavaScript Bible* by Danny Goodman (IDG), *JavaScript: The Definitive Guide* by David Flanagan (O'Reilly), and *Pure JavaScript* by R. Allen Wyke, Jason D. Gilliam, and Charlton Ting (Sams). For a free JavaScript reference, see Netscape's DevEdge Online website at http://developer.netscape.com:80/docs/manuals/javascript.html.

Formatting non-standard value types

In addition to the standard types of values that can be passed to a function, such as integer, string, and so on, Fireworks accepts additional data types for certain functions.

- Some functions take values that are Fireworks objects. These values are explained in “The Document Object Model” on page 9.
- Some functions take a string in a specific format. Others take value types that are not Fireworks objects but are JavaScript object types specific to Fireworks. These types of arguments are described below, in alphabetical order.

Color strings

Functions that take colors as arguments use the HTML syntax of "#rrggbb". You can specify a color with an alpha (transparency) component by passing a longer string of the form "#rrggbbaa".

fileURL

Many arguments that accept a file or folder name require that the name be fully qualified as a URL, in the following format:

"file:///folder1/folder2/myfile.png" && a file name, no drive spec
"file:///C|/Program Files/Macromedia/Fireworks" && a folder name, with drive spec

Note the following rules for this format.

- There are three forward slashes following the file: part of the fileURL.
- A vertical bar or pipe character (|) is used in place of a colon (:) after a drive letter.
- All slashes are forward (/).

Mask

The format for mask is {*maskBounds:* rectangle, *maskKind:* string, *maskEdgeMode:* string, *featherAmount:* int, *maskData:* hex-string}.

- *maskBounds* specifies the bounding rectangle of the mask area.
- Acceptable values for *maskKind* are "rectangle", "oval", "zlib compressed", "rle compressed", or "uncompressed".
- If *maskKind* is "*rectangle*" or "oval", the *maskData* string is ignored, and a mask of the right shape is constructed that fills *maskBounds*, and that has the edge specified by *maskEdgeMode* and *featherAmount*.
- If *maskKind* is "zlib compressed", "rle compressed", or "uncompressed", the *maskData* string is presumed to contain 8-bit mask data in hexadecimal format that precisely matches the *maskBounds* to define the mask.

Matrix

The format for a matrix is {*matrix:* [float, float, float, float, float, float, float, float, float]}. This manual assumes that you know how to use these nine values to construct a three-by-three transformation matrix; discussion of the construction of transformation matrices is beyond the scope of this manual.

Point

The format for a point is {*x:* float, *y:* float}. For example, dom.addNewLine(*startPoint, endPoint*) could look this this:

```
fw.getDocumentDOM().addNewLine({x:64.5, y:279.5}, {x:393.5, y:421.5});
```

Rectangle

The format for a rectangle is {*left:* float, *top:* float, *right:* float, *bottom:* float}. For example, dom.addNewOval(*boundingRectangle*) could look like this:

```
fw.getDocumentDOM().addNewOval({left:72, top:79, right:236, bottom:228});
```

Resolution

The format for resolution is {*pixelsPerUnit:* float, *units:* string}. Acceptable values for units are "inch" or "cm". For example, dom.setDocumentResolution*(resolution)* could look like this:

```
fw.getDocumentDOM().setDocumentResolution({pixelsPerUnit:72, units:"inch"});
```

CHAPTER 2
The Document Object Model

If you want to extend Fireworks by writing or modifying a JavaScript extensibility file, you must become familiar with the Fireworks Document Object Model, or DOM. A DOM is a tree that discloses structure in terms of objects, properties, and methods. The Fireworks DOM is structured as follows.

- At the root level of the DOM tree, there are five methods that are always available. These are described in "Global methods" on page 11.
- Also at the root level, there are five objects that are always available. These are described in "Global objects" on page 12.
- The trunk of the tree is the Fireworks object itself, described in "The Fireworks object" on page 23.
- There are a large number of objects that provide access to elements within a Fireworks document. These are described in "Fireworks document objects" on page 24.
- Finally, there is a set of objects that you can use to specify the format of HTML to use when exporting from Fireworks. These are described in "Objects for HTML export templates" on page 49.

How to use the DOM

You send calls to the DOM to determine current settings and to change settings for a Fireworks document. For example, the following command returns true if the first open document has been modified since the last time it was saved.

```
var changed = fw.documents[0].isDirty;
```

Note: Lists of objects are stored in one-dimensional arrays and are referred to by index. They are zero-based, which is why fw.documents[0] references the first open document. Similarly, a statement such as Document.layers[3] references the fourth layer in the current document.

You can also pass values for all properties that are not read-only to change elements of a document. For example, the following command sets the fifth brush in the third open document to a square shape.

```
fw.documents[2].brushes[4].shape = "square";
```

This example includes the following properties:

- documents is a property of the Fireworks object, and contains an array of Document objects.
- brushes is a property of the Document object, and contains an array of Brush objects.
- shape is a property of the Brush object.

In some cases, you can use DOM calls or API calls to perform the same functions. In other cases, a certain function may be available in either the DOM or the API, but not in both.

For example, if the first open document is the current document, the following code snippets have the same effect. (As explained in "Fireworks JavaScript API" on page 57, fw.getDocumentDOM() references the current document.)

```
fw.getDocumentDOM().setDocumentResolution({pixelsPerUnit:72, units:"inch"});
fw.documents[0].resolution =72;
fw.documents[0].resolutionUnits ="inch";
```

Note: Throughout this manual, optional arguments are enclosed in {braces}.

Global methods

The following table lists the global Fireworks methods, along with their data type and, where appropriate, acceptable values and notes.

Method	Data type	Notes
alert(*message*)	string	Displays a string in a modal alert box. An OK button is displayed. Returns nothing.
confirm(*message*)	string	Displays a string in a modal alert box. Both OK and Cancel buttons are displayed. Returns true if OK is clicked, false if Cancel is clicked.
prompt(*caption*, *text*)	string, string	Prompts the user (with the string specified by *text*) to enter a string in a modal dialog box; the dialog box is titled with the string specified by *caption*. Returns the string entered if OK is clicked, NULL if Cancel is clicked.
write(*arg1*, *arg2*, ..., *argN*)	strings	Same as WRITE_HTML. WRITE_HTML was created to let you differentiate in your code between HTML output calls from other JavaScript calls.
WRITE_HTML(*arg1*, *arg2*, ..., *argN*)	strings	Available only when exporting. Converts each argument to a string, and writes it to the HTML output file. To enter an end-of-line, use "\n"; this is converted to the correct line ending for the platform you are using. For more information, see "Objects for HTML export templates" on page 49.

Global objects

This section describes the five global objects that are always available: App, Document, Errors, Files, and Find.

Note: For information on how to format non-standard data types, such as rectangle, point, or fileURL, see "Formatting non-standard value types" on page 6.

App

The following table lists the properties and methods of the App object, along with their data type and, where appropriate, acceptable values and notes. Read-only properties are marked with a bullet (•).

Property or Method	Data type	Notes
appBatchCodeDir •	string	fileURL for the Batch Code directory.
appDir •	string	fileURL for the directory containing the Fireworks application.
appExportSettingsDir •	string	fileURL for the Export Settings directory.
appFavoritesDir •	string	fileURL for the URL Libraries directory.
appHelpDir •	string	fileURL for the Fireworks help file directory.
appHtmlCodeDir •	string	fileURL for the HTML Code directory.
appJsCommandsDir •	string	fileURL for the Commands directory.
appJsExtensionsDir •	string	fileURL for the JSExtensions directory.
appMacCreator •	string	"MKBY"
appMacJsfFileType •	string	"TEXT"
appPatternsDir •	string	fileURL for the Patterns directory.
appPresetsDir •	string	fileURL for the Presets directory.
appSettingsDir •	string	fileURL for the Settings directory.
appStylesDir •	string	fileURL for the Styles directory.
appSymbolLibrariesDir •	string	fileURL for the Libraries directory.
appTexturesDir •	string	fileURL for the Textures directory.
appXtrasDir •	string	fileURL for the Xtras directory.

Property or Method	Data type	Notes
batchStatusString	string	The string currently displayed in the Batch Progress dialog box. Set this property to change the string being displayed. Use with progressCountCurrent and progressCountTotal.
chooseScriptTarget Dialog(*formatlist*)	string	Displays a dialog box that lets the user choose the target document(s) for an operation. This dialog box allows you to specify the files currently open, the files in the project list, or files explicitly selected by the user. The arguments are similar to App.locateDocDialog(), except that the *formatlist* is required, and you cannot specify a maximum number of documents.
dismissBatchDialogWhen Done	boolean	Set this to true to force the Batch Progress dialog box to be closed automatically (without user intervention) when the script is finished running. Has no effect if the Batch Progress dialog box is not displayed.
documentList •	array	Array of the current open Document objects (see page 15). If no document is open, it returns an array of length zero.
findOpenDocument (*docname*)	fileURL	Checks to see if Fireworks already has *docname* open in a document window. If it does, returns that Document object; otherwise returns NULL.
getPref(*prefname*)	string	Returns the preference value associated with the specified preference key. A complete list of these values is beyond the scope of this manual, but the format of *prefname* exactly matches that in the Fireworks Preferences file. To set a preference value, use App.setPref().
locateDocDialog (*maxnumdocs* {,*formatlist*})	integer, string	Displays a dialog box that lets the user choose one or more files. *maxnumdocs* is the maximum number of documents to be chosen. *formatlist* is an optional list of acceptable file types to open; if omitted, all openable files are listed. The return value is an array of fileURLs, or NULL if the dialog box is canceled. For syntax details, see "Using App.locateDocDialog()" on page 15.
openDocument (*docname*)	fileURL	Opens the specified file in a new document window. If the file is already open, it is opened again; to avoid redundant opens, call findOpenDocument() first. If the file can be opened, returns that Document object. Returns NULL if the document cannot be opened.
platform •	string	The string "mac" if Fireworks is running in on the Macintosh, and "win" if running in Windows.

Property or Method	Data type	Notes
progressCountCurrent	integer	The first number (x) displayed in the Batch Progress dialog box, in the "File x of y" field. Set this property to change the number being displayed.
progressCountTotal	integer	The second number (y) displayed in the Batch Progress dialog box, in the "File x of y" field. Set this property to change the number being displayed.
quit()	none	Quits Fireworks. Prompts to save files that have changed since the last time they were saved.
setPref(*prefname*, *prefval*)	string, string	Sets the value associated with the specified preference key. A complete list of these values is beyond the scope of this manual, but the format of *prefname* and *prefval* exactly matches those in the Fireworks Preferences file. To return the value associated with a preference key, use App.getPref().

Using App.locateDocDialog()

The format for the formatlist argument is a string like the following:

"(formatname1) (formatname2) (formatname3) (formatname4) (formatname5)"

The following table lists acceptable values for formatname and the file type each value represents.

Value	File type
"ADOBE AI3"	Adobe® Illustrator®
"Fireworks JavaScript"	Fireworks JSF
"kMoaCfFormat_BMP"	bitmap
"kMoaCfFormat_FreeHand7and8"	Macromedia FreeHand® 7.0 or 8.0
"kMoaCfFormat_GIF"	GIF
"kMoaCfFormat_JPEG"	JPEG
"kMoaCfFormat_PICT"	Macintosh PICT
"kMoaCfFormat_RTF"	Rich Text
"kMoaCfFormat_Text"	plain text
"kMoaCfFormat_TIFF"	TIFF
"PNG"	PNG
"PS30"	Photoshop® PSD

Document

The following table lists the properties of the Document object, along with their data type and, where appropriate, acceptable values and notes. There are also many API calls you can use for working with documents. For more information, see "Document functions" on page 59.

Property	Data type	Notes
backgroundColor	color string in the format "#rrggbb" or "#rrggbbaa"	
backgroundUrl	string	
brushes •	array	Array of Brush objects available for use in the document (see page 24).

Property	Data type	Notes
currentFrameNum	zero-based integer	
currentLayerNum	zero-based integer	
defaultAltText	string	Default alt text for the output images. It works for single and sliced images. Sliced images all get the default, unless specific text is specified for a slice.
exportFormatOptions	object	Identical to exportOptions (below). Included for backwards compatibility with version 2 of Fireworks.
exportOptions	object	ExportOptions object (see page 35).
exportSettings	object	ExportSettings object (see page 38).
filePathForRevert	fileURL	Name of file from which this document was opened, or NULL if created from scratch.
filePathForSave	fileURL	Name of file to which this document has been saved, or NULL if never saved.
fills •	array	Array of Fill objects available for use in the document (see page 39).
frameCount	integer	
frameLoopingCount	integer	–1 — don't repeat 0 — repeat forever Other values > 0 — repeat this number of times
frames •	array	Array of Frame objects in the document (see page 39).
gammaPreview	boolean	
gradients •	array	Array of Gradient objects available for use in the document (see page 40).
gridColor	color string in the format "#rrggbb" or "#rrggbbaa"	
gridOrigin	point	
gridSize	point	gridSize.x is the horizontal grid size; gridSize.y is the vertical grid size.
guides •	object	Guides object (see page 41).
height	integer	Value in pixels.

Property	Data type	Notes
isDirty	boolean	true if the document has been modified since the last time it was saved.
isSymbolDocument •	boolean	Returns true if the document is a Symbol or Button document, false if it is a normal document. You may see this when walking through the list of open documents and one of them is a symbol-editing window.
isValid	boolean	Occasionally a document is closed, but the JavaScript object for it briefly remains active; the isValid property is useful for catching these "stale" documents.
lastExportDirectory	fileURL	
lastExportFile	fileURL	
layers •	array	Array of Layer objects in the document (see page 43).
left	integer.	Value in pixels.
mapType	string	"client", "server", "both"
matteColor	color string in the format "#rrggbb" or "#rrggbbaa"	
onionSkinAfter	integer	
onionSkinBefore	integer	
pathAttributes	object	PathAttrs object (see page 44)
pngText	object	
resolution	float	1 to 5000
resolutionUnits	string	"inch", "cm"
textures •	array	Array of Texture objects available for use in the document (see page 49)
top	integer	Value in pixels
useMatteColor	boolean	
width	integer	Value in pixels.

Errors

All Errors properties are read-only strings used to make localization of scripts easier. They return localized error messages appropriate to the error in question. For example, the English version of Fireworks returns "Memory is full." for the EOutOfMem property.

Below is an alphabetical list of the properties of the Errors object.

EAppAlreadyRunning, EAppNotSerialized, EArrayIndexOutOfBounds, EBadFileContents, EBadJsVersion, EBadNesting, EBadParam, EBadParamType, EBadSelection, EBufferTooSmall, ECharConversionFailed, EDatabaseError, EDeletingLastMasterChild, EDiskFull, EDuplicateFileName, EFileIsReadOnly, EFileNotFound, EGenericErrorOccurred, EGroupDepth, EIllegalThreadAccess, EInternalError, ELowOnMem, ENoActiveDocument, ENoFilesSelected, ENoNestedMastersOrAliases, ENoNestedPasting, ENoSliceableElems, ENoSuchElement, ENotImplemented, ENotMyType, EOutOfMem, EResourceNotFound, ESharingViolation, EUnknownReaderFormat, EUserCanceled, EUserInterrupted, EWrongType

Files

The following table lists the properties and methods of the Files object, along with their data type and, where appropriate, acceptable values and notes.

Property or Method	Data type	Notes
close()	none	Closes the file referred to by this Files object. You are not required to call this (the file is closed when the Files object is destroyed), but it is useful for controlling the access to a file.
copy(*docname1*, *docname2*)	fileURL, fileURL	Copies the file specified in the first argument to the file and directory specified in the second argument. Only files (not directories) may be copied. The files need not reside on the same drive, and the method does not overwrite a file if it already exists. Returns true if the copy is successful, false otherwise.
createDirectory(*dirname*)	fileURL	Creates the specified directory. Returns true if successful, false otherwise.
createFile(*fileURL*, {*macType* {,*macCreator*}})	fileURL, string, string	Creates the specified file in the specified location. The file must not already exist. The last two optional arguments let you specify the Macintosh file type and file creator fields. If used, the *macType* and *macCreator* strings should each be strings of exactly four characters in length.
deleteFile(*docOrDir*)	fileURL	Deletes the specified file or directory. Returns true if successful, false if the file or directory does not exist or cannot be deleted.

Property or Method	Data type	Notes
deleteFileIfExisting (*docOrDir*)	fileURL	Delete the specified file or directory. Returns true if successful, false if the file or directory cannot be deleted. Unlike deleteFile(), this method returns true if the file or directory does not exist.
enumFiles(*docOrDir*)	fileURL	Returns an array of fileURLs. If *docOrDir* is a directory, the array contains an entry for every file or directory contained in the specified directory. If *docOrDir* is a file, the array contains a single entry (the file passed in).
exists(*docOrDir*)	fileURL	Returns true if *docOrDir* refers to a directory or file that exists; false otherwise.
getDirectory(*docname*)	fileURL	Returns just the directory name from *docname*. For example, Files.getFilename("file:///work/logo.png") returns "file:///work".
getExtension(*docname*)	string	Return the filename extension, if any, of *docname*. For example, Files.getExtension("birthday.png") returns ".png". If the filename has no extension, an empty string is returned. A fileURL is acceptable here.
getFilename(*docname*)	fileURL	Returns just the filename from *docname*. For example, Files.getFilename("file:///work/logo.png") returns "logo.png".
getLastErrorString()	none	If the last call to a method in a Files object resulted in an error, returns a string describing the error. If the last call succeeded, returns NULL.
getTempFilePath ({*dirname*})	fileURL	Returns a fileURL in the Temporary Files directory or in the specified directory. This method does not create a file; it simply returns a unique fileURL that does not conflict with any existing file in the directory. If *dirname* is passed and is not NULL, the URL returned indicates a file in the specified directory rather than in the Temporary Files directory.
isDirectory(*dirname*)	fileURL	Returns true if the specified URL refers to a directory that exists; false otherwise.
makePathFromDirAndFile (*dirname, plainFilename*)	fileURL, string	Concatenates the two arguments to return a fileURL that references the specified filename in the specified directory. For example, Files.makePathFromDirAndFile("file:///work/reports", "logo.png") returns "file:///work/reports/logo.png".

Property or Method	Data type	Notes
open(*docname, writeTF*)	fileURL, boolean	Opens the specified file for reading or writing. If the second argument is true, the file is opened for writing; otherwise it is opened for reading. If the file cannot be opened, NULL is returned; otherwise a Files object is returned.
readline()	none	Reads the next line from the file referred to by the current Files object, and returns it as a string. The end-of-line character(s) are not included in the string. Returns NULL if end-of-file is reached, or if the line is more than 2048 characters long.
rename(*docname, newPlainFilename*)	fileURL, string	Renames *docname* on disk. For example, Files.rename("file:///work/logo.png", "oldlogo.png") renames "file:///work/logo.png" to "file:///work/oldlogo.png". Returns true if the rename is successful, false otherwise.
setFilename(*docname, newPlainFilename*)	fileURL, string	Returns a fileURL with *docname* replaced by *newPlainFilename*. For example, Files.setFilename("file:///work/logo.png", "oldlogo.png") returns "file:///work/oldlogo.png". This method does not affect the file on disk in any way; it simply provides a convenient way to manipulate file URLs. To change the name on disk, use rename().
swap(*docname1, docname2*)	fileURL, fileURL	Swaps the contents of the two specified files, so that each file contains the contents of the other file. Only files (not directories) may be swapped, and both files must reside on the same drive. Returns true if the swap is successful, false otherwise.
write(*textString*)	string	Writes the specified string to the file referred to by the current Files object. No end-of-line characters are appended; to include end-of-line, use "\n".

Find

There are several different ways a Find object can be specified, depending on what you want to find and replace. Use the whatToFind property to specify the type of find operation, along with the properties associated with each legal value for whatToFind. These properties are listed in the following tables. Read-only properties are marked with a bullet (•).

To find and replace text

Property or Method	Data type	Notes
whatToFind	string	"text"
find	string	Text to find.
replace	string	Text to use as replacement text.
matchCase	boolean	
wholeWord	boolean	
regExp	boolean	If true, the find and replace text is interpreted as a Regular Expression.

To find and replace fonts and styles

Property or Method	Data type	Notes
whatToFind	string	"font"
find	string	Name of font to find.
replace	string	Name of font to use as replacement.
findStyle	integer	Number that represents the style to find: AnyStyle = -1 Plain = 0 Bold = 1 Italic = 2 BoldItalic = 3 Underline = 4 BoldUnderline = 5 ItalicUnderline = 6 BoldItalicUnderline = 7
replaceStyle	integer	Number that represents the style to be used as replacement.
findMinSize	integer	0 to 9999
findMaxSize	integer	0 to 9999
replaceSize	integer	0 to 9999, or pass -1 to leave size as is

To find and replace colors, fills, strokes, and effects

Property or Method	Data type	Notes
whatToFind	string	"color"
find	color string in the format "#rrggbb" or "#rrggbbaa"	Color to find.
replace	color string in the format "#rrggbb" or "#rrggbbaa"	Color to use as replacement
fills	boolean	Set to true if you want to replace fills that match the specified colors.
strokes	boolean	Set to true if you want to replace strokes that match the specified colors.
effects	boolean	Set to true if you want to replace effects that match the specified colors.

To find and replace URLs

Property or Method	Data type	Notes
whatToFind	string	"url"
find	fileURL	URL to find.
replace	fileURL	URL to use as replacement text.
wholeWord	boolean	
matchCase	boolean	
regExp	boolean	If true, the find and replace text is interpreted as a Regular Expression.

The Fireworks object

The Fireworks object is the global object, which you can use to set or retrieve properties relating to the current operating environment. For additional environment properties, see "App" on page 12.

The following table lists the properties and methods of the Fireworks object, along with their data type and, where appropriate, acceptable values and notes. Read-only properties are marked with a bullet (•).

Note: For information on how to format non-standard data types, such as rectangle, point, or fileURL, see "Formatting non-standard value types" on page 6.

Refer to Fireworks objects by using fw.propertyName or fireworks.propertyName. Note that fireworks must be lower case.

Property or Method	Data type	Notes
dismissBatchDialogWhenDone	boolean	Largely obsolete, used for backwards compatibility with version 2 of Fireworks.
documents •	array	Array of the current open Document objects (see page 15). If no document is open, it returns an array of length zero.
historyPalette •	object	History panel object. There are no DOM properties for the History panel, only API calls. For more information, see *"History panel functions" on page 123*.
screenRect •	rectangle	The size of the main screen on this computer, in pixels. Useful for positioning windows or panels.
selection	array	Array of the selected objects in the active document. If nothing is selected. returns an array of length zero. If no document is open, returns NULL.
styles •	array	Array of the Style objects (see page 46) currently loaded in the Style panel.

Fireworks document objects

This section describes the objects that provide access to elements within a Fireworks document.

Note: For information on how to format non-standard data types, such as rectangle, point, or fileURL, see "Formatting non-standard value types" on page 6.

Behavior

The following table lists the properties of the Behavior object, along with their data type and, where appropriate, acceptable values and notes.

Property	Data type	Notes
call	string	The JavaScript call for the behavior. For legal values, see "Using the addBehavior() function" on page 136
event	string	"onMouseOver", "onClick", "onMouseOut", "onLoad", "**ANY**" (the **ANY** argument is used as a wildcard value in some situations)

Brush

The following table lists the properties of the Brush object, along with their data type and, where appropriate, acceptable values and notes.

Property	Data type	Notes
alphaRemap	string	"none", "white neon", "harsh wet", "smooth neon", "wavy gravy", "white neon edge"
angle	integer	0 to 360
antiAliased	boolean	
aspect	float	0 to 100
blackness	float	0 to 100
category	string	
concentration	float	0 to 100
diameter	integer	0 to 1000
feedback	string	"none", "brush", "background"
flowRate	float	0 to 100

Property	Data type	Notes
maxCount	integer	0 to 64
minSize	float	0 to 100
name	string	
sensitivity_x_y	integer	0 to 100 where x is one of: pressure, speed, hDir, vDir, random; and y is one of: size, angle, opacity, blackness, scatter, hue, lightness, saturation. For example, sensitivity_pressure_size
shape	string	"circle", "square"
softenMode	string	"bell curve", "linear"
softness	float	0 to 100
spacing	float	0 to 500 (a percentage, up to 500%)
textureBlend	float	0 to 100
textureEdge	float	0 to 100
tipColoring	string	"random", "uniform", "complementary", "hue", "shadow"
tipCount	integer	1 to 32
tipSpacing	float	0 to 100
tipSpacingMode	string	"random", "diagonal", "circular"
type	string	"natural", "simple"

Contour

The following table lists the properties of the Contour object, along with their data type and, where appropriate, acceptable values and notes.

Property	Data type	Notes
isClosed	boolean	
nodes	array	Array of ContourNode objects on the contour (see the next heading)

ContourNode

The following table lists the properties of the ContourNode object, along with their data type and, where appropriate, acceptable values and notes.

Property	Data type	Notes
dynamicInfo	array	Array of ContourNodeDynamicInfo objects (see the next heading) on this ContourNode object.
isCurvePoint	boolean	
isSelectedPoint	boolean	
predX	float	The x coordinate of the contour node's preceding control point.
predY	float	The y coordinate of the contour node's preceding control point.
randomSeed	integer	0 to 65535
succX	float	The x coordinate of the contour node's following control point.
succY	float	The y coordinate of the contour node's following control point.
x	float	The x coordinate of the contour node's main control point.
y	float	The y coordinate of the contour node's main control point.

ContourNodeDynamicInfo

The following table lists the properties of the ContourNodeDynamicInfo object, along with their data type and, where appropriate, acceptable values and notes.

Property	Data type	Notes
duration	float	0.0 to 65535.0 milliseconds
pressure	float	0.0 to 1.0
velocity	float	0.0 to 255.9999 pixels-per-millisecond

Effect

Each Effect object has a different set of properties because every effect has different attributes that can be set. The properties for various Effect objects are listed in the following tables, in alphabetical order.

Note: In addition to the properties listed, each Effect object has two optional string properties: category and name.

Bevel

Use the BevelType property of this Effect to set a bevel as inner, outer, raised embossed, inset embossed, or glow effect.

Property	Data type	Notes
EffectMoaID	string	"{7fe61102-6ce2-11d1-8c76000502701850}"
EffectIsVisible	boolean	Allows for an effect to be included but temporarily hidden. Default value is true.
AngleSoftness	integer	
BevelContrast	integer	0 to 100 percent
BevelType	integer	InnerBevel = 0; OuterBevel = 1; RaiseEmboss = 2; InsetEmboss = 3; GlowEffect = 4
BevelWidth	integer	
ButtonState	integer	BevelButtonUp = 0; BevelButtonOver = 1; BevelButtonDown = 2; BevelButtonHit = 3
DownBlendColor	color string in the format "#rrggbb" or "#rrggbbaa"	Color that is blended on top of graphic if ButtonState = 2 (BevelButtonDown).
EdgeThreshold	integer	Controls the opacity at which the edge of the effect is defined. Use 1 if BevelType = 4 (for GlowEffect); otherwise, use 0.
EmbossFaceColor	color string in the format "#rrggbb" or "#rrggbbaa"	Color that is blended onto the face of the object when embossing.
GlowStartDistance	integer	
GlowWidth	integer	

Property	Data type	Notes
HiliteColor	color string in the format "#rrggbb" or "#rrggbbaa"	The color blended to provide the specular lighting type effect. Used by beveling only. Currently white is always used for internally created effects (though any value should work).
HitBlendColor	color string in the format "#rrggbb" or "#rrggbbaa"	Color that is blended on face of the graphic if ButtonState = 3 (BevelButtonHit).
LightAngle	integer	
LightDistance	integer	
MaskSoftness	integer	
OuterBevelColor	color string in the format "#rrggbb" or "#rrggbbaa"	
ShadowColor	color string in the format "#rrggbb" or "#rrggbbaa"	
SlopeMultiplier	float	A multiplier used to calculate the magnitude of the bevel slope. Default effects all use 1, but other values should work. For example, 0.5 gives a more subtle slope while 2.0 gives a sharper slope.
SlopeType	integer	flat slope = 0; smooth slope = 1; inverted smooth slope = 2; frame 1 slope = 3; frame 2 slope = 4; ring slope = 5; ruffle slope = 6

Blur

Property	Data type	Notes
EffectMoaID	string	"{f1cfce41-718e-11d1-8c8200a024cdc039}"
EffectIsVisible	boolean	Allows for an effect to be included but temporarily hidden. Default value is true.

Blur More

Property	Data type	Notes
EffectMoaID	string	"{f1cfce42-718e-11d1-8c8200a024cdc039}"
EffectIsVisible	boolean	Allows for an effect to be included but temporarily hidden. Default value is true.

Brightness/Contrast

Property	Data type	Notes
EffectMoaID	string	"{3439b08c-1921-11d3-9bde00e02910d580}"
EffectIsVisible	boolean	Allows for an effect to be included but temporarily hidden. Default value is true.
brightness_amount	long	-100 to 100
contrast_amount	long	-100 to 100

Convert to Alpha

Property	Data type	Notes
EffectMoaID	string	"{2932d5a2-ca48-11d1-8561000502701850}"
EffectIsVisible	boolean	Allows for an effect to be included but temporarily hidden. Default value is true.

Curves

Property	Data type	Notes
EffectMoaID	string	"{3439b08e-1923-11d3"-9bde00e02910d580}"
EffectIsVisible	boolean	Allows for an effect to be included but temporarily hidden. Default value is true.
rgb_points	vector of points	Each of these properties is a vector of points where x = input level and y = output level. All x and y values must be between 0 and 255, and the points must be sorted in ascending order of x coordinate.
red_points		
green_points		
blue_points		

Drop Shadow

Property	Data type	Notes
EffectMoaID	string	"{a7944db8-6ce2-11d1-8c76000502701850}"
EffectIsVisible	boolean	Allows for an effect to be included but temporarily hidden. Default value is true.
ShadowAngle	float	
ShadowBlur	float	
ShadowColor	color string in the format "#rrggbb" or "#rrggbbaa"	
ShadowDistance	float	
ShadowType	integer	0 = normal shadow, 1 = knockout shadow

Find Edges

Property	Data type	Notes
EffectMoaID	string	"{fc7093f1-f95c-11d0-8be200a024cdc039}"
EffectIsVisible	boolean	Allows for an effect to be included but temporarily hidden. Default value is true.

Gaussian Blur

Property	Data type	Notes
EffectMoaID	string	"{d04ef8c0-71b3-11d1-8c8200a024cdc039}"
EffectIsVisible	boolean	Allows for an effect to be included but temporarily hidden. Default value is true.
gaussian_blur_radius	float	0.1 to 250

Hue/Saturation

Property	Data type	Notes
EffectMoaID	string	"{3439b08d-1922-11d3-9bde00e02910d580}"
EffectIsVisible	boolean	Allows for an effect to be included but temporarily hidden. Default value is true.
hue_amount	long	-180 to 180 if hls_colorize is false; 0 to 360 if hls_colorize is true.
saturation_amount	long	-100 to 100 if hls_colorize is false; 0 to 100 if hls_colorize is true.
lightness_amount	long	0 to 100
hls_colorize	boolean	

Inner Shadow

Property	Data type	Notes
EffectMoaID	string	"{5600f702-774c-11d3-baad0000861f4d01}"
EffectIsVisible	boolean	Allows for an effect to be included but temporarily hidden. Default value is true.
ShadowAngle	integer	
ShadowBlur	integer	
ShadowColor	color string in the format "#rrggbb" or "#rrggbbaa"	
ShadowDistance	integer	
ShadowType	integer	0 = normal shadow, 1 = knockout shadow

Invert

Property	Data type	Notes
EffectMoaID	string	"{d2541291-70d6-11d1-8c8000a024cdc039}"
EffectIsVisible	boolean	Allows for an effect to be included but temporarily hidden. Default value is true.

Levels

Property	Data type	Notes
EffectMoaID	string	"{d04ef8c1-71b4-11d1-8c8200a024cdc039}"
EffectIsVisible	boolean	Allows for an effect to be included but temporarily hidden. Default value is true.
source_low_rgb	long	These values are all input levels to the filter, with values of 0 to 255.
source_high_rgb		
source_low_red		
source_high_red		
source_low_green		
source_high_green		
source_low_blue		
source_high_blue		
dest_low_rgb	long	These values are all output levels to the filter, with values of 0 to 255.
dest_high_rgb		
dest_low_red		
dest_high_red		
dest_low_green		
dest_high_green		
dest_low_blue		
dest_high_blue		
gamma_rgb	float	These values are all gamma levels to the filter, with values of 0.1 to 10.0.
gamma_red		
gamma_green		
gamma_blue		

Sharpen

Property	Data type	Notes
EffectMoaID	string	"{c20952b1-fc76-11d0-8be700a024cdc039}"
EffectIsVisible	boolean	Allows for an effect to be included but temporarily hidden. Default value is true.

Sharpen More

Property	Data type	Notes
EffectMoaID	string	"{1f2f2591-9db7-11d1-8cad00a024cdc039}"
EffectIsVisible	boolean	Allows for an effect to be included but temporarily hidden. Default value is true.

Unsharp Mask

Property	Data type	Notes
EffectMoaID	string	"{f1cfce44-718e-11d1-8c8200a024cdc039"
EffectIsVisible	boolean	Allows for an effect to be included but temporarily hidden. Default value is true.
unsharp_mask_radius	float	0.1 to 250
unsharp_mask_amount	long	1 to 500
unsharp_mask_threshold	long	0 to 255

EffectList

The following table lists the properties of the EffectList object, along with their data type and, where appropriate, acceptable values and notes.

Property	Data type	Notes
category	string	
effects	array	Array of Effect objects (see page 27).
name	string	

Element

Element is an Abstract Base Class; nothing of class Element ever exists. However, it is useful for simplifying the other class descriptions. Read-only properties are marked with a bullet (•).

Property	Data type	Notes
blendMode	string	"normal", "multiply", "screen", "darken", "lighten", "difference", "hue", "saturation", "color", "luminosity", "invert", "tint", "erase"
effectList	object	EffectList object (see page 33).
height •	float	Read-only in the base class; other properties or API calls are used to resize specific types of elements.
left	float	May round to an integer.
opacity	float	Acceptable values are 0 to 100, and represent percent opacity.
top	float	May round to an integer.
visible	boolean	
width •	float	Read-only in the base class; other properties or API calls are used to resize specific types of elements.

ExportFrameInfo

The following table lists the properties of the ExportFrameInfo object, along with their data type and, where appropriate, acceptable values and notes.

Property	Data type	Notes
gifDisposalMethod	string	"unspecified", "none", "background", "previous"
delayTime	integer	
frameHidden	boolean	

ExportOptions

Note: When using this object to set properties, the only required property is exportFormat. If other properties are not specified, their default values are used.

The following table lists the properties of the ExportOptions object, along with their data type and, where appropriate, acceptable values and notes.

In addition, use the following information to understand the rules for determining scaling in this object.

If useScale is true, then percentScale is used to uniformly scale the export, regardless of the setting of applyScale.

If useScale is false and applyScale is false, no scaling is done on export.

If useScale is false and applyScale is true, then xSize and ySize determine scaling as follows:

- If the value is positive, it means "this is exactly the size for this axis".
- If the value is zero, it means "this axis varies without limit".
- If the value is negative, it means "this axis varies, but may be no larger than abs(value)".

If one value is positive and one is negative, the positive value is always used. Thus, this gives the following possibilities:

- xSize < 0, ySize < 0 – use min(xSize, ySize) scaling
- xSize < 0, ySize = 0 – use xSize scaling
- xSize < 0, ySize > 0 – use ySize scaling
- xSize = 0, ySize < 0 – use ySize scaling
- xSize = 0, ySize = 0 – illegal, use scale of 1.0
- xSize = 0, ySize > 0 – use ySize scaling
- xSize > 0, ySize < 0 – use xSize scaling
- xSize > 0, ySize = 0 – use xSize scaling
- xSize > 0, ySize > 0 – do not use; use useScale = true and percentScale = 0 to 100 instead

Property	Data type	Notes
animAutoCrop	boolean	Default is true.
animAutoDifference	boolean	Default is true.
applyScale	boolean	Default is false.

Property	Data type	Notes
colorMode	string	"indexed", "24 bit", "32 bit"; default is "indexed".
crop	boolean	Default is false.
cropBottom	integer	Default is 0.
cropLeft	integer	Default is 0.
cropRight	integer	Default is 0.
cropTop	integer	Default is 0.
ditherMode	string	"none", "diffusion", "2 by 2"; default is "none".
ditherPercent	integer	0 to 100; default is 100.
exportFormat	string	"GIF", "JPEG", "PNG", "custom", "GIF animation"; no default — this value must be specified.
frameInfo	array	Array of ExportFrameInfo objects (see page 34); can be NULL; default is NULL.
interlacedGIF	boolean	Default is false.
jpegQuality	integer	1 to 100; default is 80.
jpegSmoothness	integer	0 to 8; default is 0.
jpegSubsampling	integer	0 to 4; default is 1.
localAdaptive	boolean	default is true.
lossyGifAmount	integer	0 to 100; default is 0.
macFileCreator	string	Default is "" (an empty string).
macFileType	string	Default is "" (an empty string).
name	string	Default is "" (an empty string).
numCustomEntries	integer	0 to 256; default is 0.
numEntriesRequested	integer	0 to 256; default is 128.
numGridEntries	integer	0 to 256; default is 6.
optimized	boolean	Default is true.
paletteEntries	array	Array of color strings (see page 6); default is NULL.
paletteInfo	array	Array of ExportPaletteInfo objects (see page 37), or NULL if all entries in the array are default values; default is NULL.

Property	Data type	Notes
paletteMode	string	"custom", "adaptive", "grid", "monochrome", "Macintosh", "Windows", "exact", "Web 216"; default is "adaptive".
paletteTransparencyType	string	"none", "index", "index alpha", "rgba"; default is "none".
percentScale	integer	1 to 100000; default is 100.
progressiveJPEG	boolean	Default is false.
savedAnimationRepeat	integer	Default is 0.
sorting	string	"none", "luminance", "popularity"; default is "none".
transparencyIndex	zero-based integer	-1 to 255; pass -1 to use the background color's index; default is –1.
useScale	boolean	Default is true.
webSnapAdaptive	boolean	Default is true.
webSnapTolerance	integer	Default is 14.
xSize	integer	-100000 to 100000; default is 0.
ySize	integer	-100000 to 100000; default is 0. See information preceding table for details on using xSize and ySize.

ExportPaletteInfo

The following table lists the properties of the ExportPaletteInfo object, along with their data type and, where appropriate, acceptable values and notes.

Property	Data type	Notes
colorLocked	boolean	
colorModified	boolean	
colorSelected	boolean	
colorTransparent	boolean	
newColorValue	color string in the format "#rrggbb" or "#rrggbbaa"	

ExportSettings

The following table lists the properties of the ExportSettings object, along with their data type and, where appropriate, acceptable values and notes.

Property	Data type	Notes
discardUnspecifiedSlices	boolean	
generateDemoHtml	boolean	
htmlDestination	string	"same", "one up", "custom", "clipboard"
setByUser	boolean	
shimGeneration	string	"none", "transparent", "internal", "nested tables"
sliceAlongGuides	boolean	
sliceAutoNaming	string	"basename_row_col", "basename_alphabetical", "basename_numeric", "row_col_basename", "alphabetical_basename", "numeric_basename"
sliceUsingUrls	boolean	
targetFrames	boolean	
templateName	string	

Fill

The following table lists the properties of the Fill object, along with their data type and, where appropriate, acceptable values and notes.

Property	Data type	Notes
category	string	
ditherColors	array	Array of two color strings (see page 6).
edgeType	string	"hard", "antialiased"
feather	integer	0 to 1000, representing the feathering value in pixels (0 means no feathering).
gradient	object	Gradient object (see page 40).
name	string	
pattern	object	Pattern object (see page 44).
shape	string	"solid", "linear", "radial", "conical", "satin", "pinch", "folds", "elliptical", "rectangular", "bars", "ripple", "waves", "pattern", "web dither"
stampingMode	string	"blend", "blend opaque"
textureBlend	float	0 to 100
webDitherTransparent	boolean	

Frame

The following table lists the properties of the Frame object, along with their data type and, where appropriate, acceptable values and notes. Read-only properties are marked with a bullet (•).

Property	Data type	Notes
layers •	array	Array of FrameNLayerIntersection objects in the document (see the next heading).
delay	integer	Hundredths of a second.
disposal	string	"unspecified", "none", "background", "previous"
visible	boolean	

FrameNLayerIntersection

The following table lists the properties of the FrameNLayerIntersection object, along with their data type and, where appropriate, acceptable values and notes. Read-only properties are marked with a bullet (•).

Property	Data type	Notes
elements •	array	Array of Element objects (see page 34).
locked	boolean	
visible	boolean	

Gradient

The following table lists the properties of the Gradient object, along with their data type and, where appropriate, acceptable values and notes.

Property	Data type	Notes
name	string	
nodes	array	Array of GradientNode objects (see the next heading) on this Gradient object.

GradientNode

The following table lists the properties of the GradientNode object, along with their data type and, where appropriate, acceptable values and notes.

Property	Data type	Notes
color	color string in the format "#rrggbb" or "#rrggbbaa"	
position	float	0.0 to 1.0

Group

Group is a subclass of the base class Element (see page 34), and contains the following properties in addition to those in Element.

Property	Data type	Notes
elements	array	Array of Element objects in the group (see page 34).
groupType	string	"normal", "mask to image", "mask to path"

Guides

The following table lists the properties of the Guides object, along with their data type and, where appropriate, acceptable values and notes.

Property	Data type	Notes
color	color string in the format "#rrggbb" or "#rrggbbaa"	
hGuides	array	Array of floating-point numbers specifying horizontal guide locations.
locked	boolean	
vGuides	array	Array of floating-point numbers specifying vertical guide locations.

Hotspot

Hotspot is a subclass of the base class Element (see page 34), and contains the following properties in addition to those in Element.

Property	Data type	Notes
altText	string	
behaviors	array	Array of Behavior objects for the hotspot (see page 24).
color	color string in the format "#rrggbb" or "#rrggbbaa"	
contour	object	Contour object for the hotspot (see page 25); used only if shape="polyline"; otherwise NULL.
shape	string	"rectangle", "circle", "polyline"
targetText	string	
urlText	string	

Image

Image is a subclass of the base class Element (see page 34). It contains no properties or methods of its own in addition to those in Element.

Instance

Instance is a subclass of the base class Element (see page 34), and contains the following properties in addition to those in Element. Read-only properties are marked with a bullet (•).

Property	Data type	Notes
symbolID •	string	An arbitrary string used to uniquely identify the symbol owning this instance.
transformMode	string	"paths", "pixels"

Layer

The following table lists the properties of the Layer object, along with their data type and, where appropriate, acceptable values and notes. Read-only properties are marked with a bullet (•).

Property	Data type	Notes
frames •	array	array of FrameNLayerIntersection objects (see page 40)
name	string	
sharing	string	"shared", "not shared"
layerType •	string	"normal", "web"

Path

Path is a subclass of the base class Element (see page 34), and contains the following properties in addition to those in Element.

Property	Data type	Notes
randSeed	float	Actually a 32-bit integer, but JavaScript integers hold only31-bit numbers, so it is stored as a floating point number.
textureOffset	point	
pathAttributes	object	PathAttrs object (see page 44).
contours	array	Array of Contour objects (see page 25) on this Path object.

PathAttrs

The following table lists the properties of the PathAttrs object, along with their data type and, where appropriate, acceptable values and notes.

Property	Data type	Notes
brush	object	Brush object (see page 24).
brushColor	color string in the format "#rrggbb" or "#rrggbbaa"	
brushPlacement	string	"inside", "center", "outside"
brushTexture	object	Texture object (see page 49).
fill	object	Fill object (see page 39).
fillColor	color string in the format "#rrggbb" or "#rrggbbaa"	
fillHandle1	point	
fillHandle2	point	
fillHandle3	point	
fillOnTop	boolean	
fillTexture	object	Texture object (see page 49).

Pattern

The following table lists the properties of the Pattern object, along with their data type and, where appropriate, acceptable values and notes.

Property	Data type	Notes
name	string	

SingleTextRun

The following table lists the properties of the SingleTextRun object, along with their data type and, where appropriate, acceptable values and notes.

Property	Data type	Notes
changedAttrs	object	TextAttrs object (see page 48).
characters	string	

SliceHotspot

SliceHotspot is a subclass of the base class Hotspot (see page 42), and contains the following properties in addition to those in Hotspot. Read-only properties are marked with a bullet (•).

Property	Data type	Notes
baseName	string	Base name for slice filenames, or NULL for auto-name.
exportOptions	object	ExportOptions object (see page 35); NULL if using current document defaults.
htmlText	string	
sliceID •	string	An arbitrary string used to uniquely identify this slice.
sliceKind	string	"image", "empty"

Style

The following table lists the properties of the Style object, along with their data type and, where appropriate, acceptable values and notes. All Style properties are read-only.

Property (read-only)	Data type	Notes
effectList	object	EffectList object (see page 33).
name	string	
pathAttributes	object	PathAttrs object (see page 44).
textBold	boolean	
textFont	string	
textItalic	boolean	
textSize	string	String of the form "#pt", where # is a numeric value.
textUnderline	boolean	
use_brush	boolean	
use_brushColor	boolean	
use_effectList	boolean	
use_fill	boolean	
use_fillColor	boolean	
use_textFont	boolean	
use_textSize	boolean	
use_textStyles	boolean	

Text

Text is a subclass of the base class Element (see page 34), and contains the following properties in addition to those in Element.

Property	Data type	Notes
antiAliased	boolean	
antiAliasMode	string	"smooth", "crisp", or "strong". This value is ignored if the antiAliased property is false.
autoKern	boolean	
orientation	string	"horizontal left to right", "vertical right to left", "horizontal right to left", "vertical left to right"
pathAttributes	object	PathAttrs object (see page 44).
randSeed	float	Actually a 32-bit integer, but JavaScript integers hold only31-bit numbers, so it is stored as a floating point number.
textRuns	object	TextRuns object (see page 48)
textureOffset	point	
transformMode	string	"paths", "pixels"

TextAttrs

The following table lists the properties of the TextAttrs object, along with their data type and, where appropriate, acceptable values and notes.

Property	Data type	Notes
alignment	string	"left", "center", "right", "justify", "stretch"
baselineShift	float	
bold	boolean	
face	string	
fillColor	color string in the format "#rrggbb" or "#rrggbbaa"	
horizontalScale	float	
italic	boolean	
kerning	float	
leading	float	
leadingMode	string	"percentage"
rangeKerning	float	
size	string	String of the form "#pt", where # is a numeric value.
underline	boolean	

TextRuns

The following table lists the properties of the TextRuns object, along with their data type and, where appropriate, acceptable values and notes.

Property	Data type	Notes
initialAttrs	object	TextAttrs object (see page 48).
textRuns	array	Array of SingleTextRun objects (see page 45) on this TextRuns object.

Texture

Texture is a subclass of the base class Element (see page 34), and contains the following read-only property in addition to those in Element.

Property (read-only)	Data type	Notes
name	string	

Objects for HTML export templates

Fireworks provides several object types that support the output of HTML and sliced images from Fireworks. These objects let you write JavaScript scripts that create templates to output the "flavor" of HTML that suits your specific requirements: generic HTML, Dreamweaver-compatible HTML, and so on. For each HTML template, you use a Slices.htt file that generates the HTML for that particular template. For more information, refer to the Slices.htt and Metafile.htt files that are installed with Fireworks.

Note: For information on how to format non-standard data types, such as rectangle, point, or fileURL, see "Formatting non-standard value types" on page 6.

BehaviorInfo

The BehaviorInfo object describes a behavior assigned to an element. There are six behaviors: Status Message, Swap Image, Button Down, Swap Image Restore, Button Highlight, and Button Restore. The following table lists the properties of the BehaviorInfo object, along with their data type and, where appropriate, acceptable values and notes. All BehaviorInfo properties are read-only.

Property (read-only)	Data type	Notes

action	integer	Specifies the type of behavior: 1 is Swap Image, 2 is Status Message, 4 is Button Down, 5 is Swap Image Restore, 6 is Button Highlight, and 7 is Button Restore. In the standard templates, these values are defined:
		var kActionStatusMessage = 1;
		var kActionSwapImage = 2;
		var kActionButtonDown = 4;
		var kActionSwapImageRestore = 5;
		var kActionButtonHighlight = 6;
		var kActionButtonRestore = 7;

downHighlight	boolean	For Button Highlight behaviors, true if there is a down highlight image.

Property (read-only)	Data type	Notes

event	integer	Specifies the type of event: 0 is Mouse Over, 1 is On Click, 2 is Mouse Out, and 3 is On Load. In the standard templates, these values are defined:
		var kEventMouseOver = 0;
		var kEventOnClick = 1;
		var kEventMouseOut = 2;
		var kEventOnLoad=3;

hasHref	boolean	For Swap Image behaviors, true if the swap image swaps in an external file. The Swap Image behavior can also swap in from a Fireworks frame.
hasStatusText	boolean	For Status Message behaviors, true if the status text is not empty.
hasTargetFrame	boolean	For Swap Image behaviors, true if the swap image swaps in another frame in the Fireworks file. The Swap Image behavior can also swap in from an external image. hasTargetFrame is always the opposite of hasHref; you cannot swap from two sources.
href	fileURL	For Swap Image behaviors, the fileURL for an external swap image file.
preload	boolean	For Swap Image behaviors, true if the image is to be preloaded.
restoreOnMouseout	boolean	If true, the original image for a SwapImage behavior is restored on mouse out.
statusText	string	For Status Message behaviors, the status message text.
targetColumnNum	zero-based integer	For Swap Image behaviors, the column in the slices table that is swapped.
targetFrameNum	zero-based integer	For Swap Image behaviors, if hasTargetFrame is true, this is the frame number that is swapped.
targetRowNum	zero-based integer	For Swap Image behaviors, the row in the slices table that is swapped.

BehaviorsList

The BehaviorsList object is an array of BehaviorInfo objects (see page 49) that describe the behaviors in an image map. The BehaviorsList object does not occur by itself. That is, all occurrences of BehaviorsList objects are members of other objects. In the following example,

```
var curBehavior = slices[i][j].behaviors[k];
```

behaviors is an object of type BehaviorsList, and curBehavior is an object of type BehaviorInfo.

The BehaviorsList object has only one property, shown in the following table.

Property (read-only)	Data type	Notes
numberOfBehaviors	integer	The number of BehaviorInfo objects (see page 49) in the BehaviorsList array (0 or more).

exportDoc

The following table lists the properties of the exportDoc object, along with their data type and, where appropriate, acceptable values and notes. All exportDoc properties are read-only.

Note: This object type does not start with a capital letter.

Property (read-only)	Data type	Notes
altText	string	String that is the alt text description for the Fireworks document.
backgroundColor	hex string	String specifying the hex color for the document canvas, without the # character; for example, "FF0000" for red background.
backgroundIsTransparent	boolean	true if the Fireworks canvas color is transparent, or if the export settings are a transparent GIF format.
backgroundLink	fileURL	String that is the background URL.
clientMap	boolean	true if client side image map was selected for the document.
filename	string	Simple URL for the exported image, relative to the HTML output; for example, "images/Button.gif". In Slices.htt, it is the base image name plus the base extension. Unless there is only one slice, Slices.htt produces actual file names of "Button_r2_c2.gif".

Property (read-only)	Data type	Notes
generateHeader	boolean	true if an HTML file is generated, false if the output goes to the Clipboard.
hasAltText	boolean	true if the Fireworks document has an alt text description.
hasBackgroundLink	boolean	true is the Fireworks document has a background URL.
height	integer	Height of the exported image in pixels. In Slices.htt, it is the total height of the output images.
htmlOutputPath	fileURL	fileURL that the HTML is being written to, including file name; for example, "file:///C|/top/nav/navbar.htm".
imagename	string	Base image name, without extension; for example, "Button".
numFrames	integer	Number of frames being exported from the Fireworks document. Not zero-based; value is 1 or more.
pathBase	string	Filename with the extension removed; for example, "images/Button".
pathSuffix	string	Extension for filename, including a period; for example, ".gif".
serverMap	boolean	true if server side image map was selected for the document.
startColumn	integer	Used only in Metafile.htt, for generating HTML for one slice. Indicates the column of the slice.
startRow	integer	Used only in Metafile.htt, for generating HTML for one slice. Indicates the row of the slice.
width	integer	Width of the export image in pixels. In Slices.htt, it is the total width of the output images.

ImageMap

The following table lists the properties and methods of the ImageMap object, along with their data type and, where appropriate, acceptable values and notes. All ImageMap properties are read-only.

Property (read-only) or Method	Data type	Notes
altText	string	The alt text description for this slice, if any.
behaviors	object	BehaviorsList object (see page 51) containing the behaviors for this slice.
hasAltText	boolean	true if the slice has an alt text description.
hasHref	boolean	true if the slice has a URL.
hasTargetText	boolean	true if the target text is not empty.
href	fileURL	The URL link for this slice.
numCoords	integer	Number of coordinates in the area. A circle always has 1 (the center), a rectangle has 2 (top left and bottom right), and a polygon has one or more.
radius	integer	Radius of the area, if shape is "circle".
shape	string	"circle", "poly", "rect"
targetText	string	Target text for this image, if any.

xCoord(*index*)	zero-based integer	Returns the x coordinate for the specified point, in pixels. For example, the following commands return the coordinates for the first point:
		var x = imagemap.xCoord(0);
		var y = imagemap.yCoord(0);
		It is possible to have negative values if the imagemap area is drawn such that it crosses the left or top sides of the image (or sliced image).

yCoord(*index*)	zero-based integer	Returns the y coordinate for the specified point, in pixels. See xCoord().

ImagemapList

The ImagemapList is an array of ImageMap objects (see the previous heading) that describe the areas in an image map. The ImageMap objects are accessed by array, as shown below, and have only one property, shown in the following table.

```
var curImagemap = imagemapList[i];
```

Property (read-only)	Data type	Notes
numberOfURLs	integer	The number of imagemap areas in the imagemap list (0 or more).

SliceInfo

The following table lists the properties and methods of the SliceInfo object, along with their data type and, where appropriate, acceptable values and notes. All SliceInfo properties are read-only.

Property (read-only) or Method	Data type	Notes
altText	string	The alt text description for this slice.
behaviors	object	BehaviorsList object (see page 51) containing the behaviors for this slice.
cellHeight	integer	Height of this table row in pixels.
cellWidth	integer	Width of this table column in pixels.
downIndex	zero-based integer	The index for this slice as a button if it is a multiple file export down button.
getFrameFileName (*frameIndex*)	zero-based integer	Returns a string that is the file name for the slice on the specified frame, without directory or extension information. For example, when exporting a file base named Button, Slices[0][0].getFrameFileName(0) returns "Button_r1_c1". Generally all slices that have images have a frame file name. For frames 1 and up, only slices that are rollovers or that are targeted by a swap image have names.
hasAltText	boolean	true if the slice has an alt text description.
hasHref	boolean	true if the slice has a URL.
hasHtmlText	boolean	true if the cell is a text only slice.
hasImage	boolean	true if this cell has an image. For text only slices, this is false.

Property (read-only) or Method	Data type	Notes
hasImagemap	boolean	true if there are imagemap hotspots in this image slice.
hasTargetText	boolean	true if the target text is not empty.
height	integer	Height of the image in pixels, including row spans.
href	fileURL	The URL link for this slice.
htmlText	string	Text for a text only slice.
imagemap	object	ImagemapList object (see page 54) containing the image map information for this slice.
imageSuffix	string	Extension for the image in this cell, including a period; for example, ".gif".
isUndefined	boolean	true if the slice does not have a slice object drawn over it. If you draw two slices that don't cover your document, Fireworks automatically generates slices to cover the rest of the document. These slices are the undefined slices.
left	integer	Left side of the cell in pixels. The left starts at 0.
nestedTableSlices	object	Slices object (see page 56) that describes a nested table that occupies the current table cell. NULL if the cell does not contain a nested table.
setFrameFileName (*frameIndex*)	zero-based integer	Sets the file name for the slice on the specified frame, without directory or extension information. You can stop an image from exporting by setting its name to "" (an empty string).
skipCell	boolean	true if this cell in the table is covered by a previous row span or column span.
targetText	string	Target text for this image, if any.
top	integer	Top of the cell in pixels. The top starts at 0.
width	integer	Width of the image in pixels, including column spans.

Slices

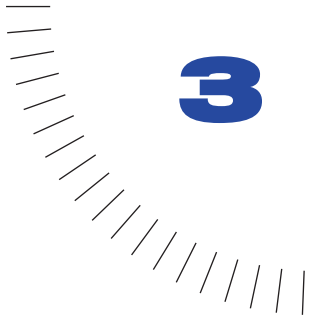
Slices is an object that has some properties, and is also a two-dimensional array of SliceInfo objects (see page 54). For example, Slices[0][0] is the SliceInfo for the first cell at row 0, column 0. The first array is rows, the second columns.

A fairly common way to access the table is:

```
var curRow;
var curCol;
for (curRow = 0; curRow<slices.numRows; curRow++) {
   for (curCol=0; curCol<slices.numColumns; curCol++) {
      var curSlice = slices[curRow][curCol]; // curSlice is the slice info for the cell at this row &
column.
      // do whatever processing with curSlice.
   }
}
```

The following table lists the properties of the Slices object, along with their data type and, where appropriate, acceptable values and notes. All Slices properties are read-only.

Property (read-only)	Data type	Notes
demoIndex	zero-based integer	Index for each file generated for multiple button file export.
doDemoHTML	boolean	true for multiple file button rollover export.
doShimEdges	boolean	true if in Document Properties, Table Shims are set to Transparent Image.
doShimInternal	boolean	true if in Document Properties, Table Shims are set to Shims From Image.
doSkipUndefined	boolean	true if in Document Properties, Export Undefined Slices is NOT checked.
imagesDirPath	string	Relative URL to the images folder. Example: "images/", or "../site_images", or "" (an empty string) if the images and the HTML are in the same directory.
numColumns	integer	Number of columns that are present in the HTML table. Does not include shim column.
numRows	integer	Number of rows that are present in the HTML table. Does not include shim row.
shimPath	string	Relative URL to the shim GIF file; for example, "images/shim.gif".



CHAPTER 3
Fireworks JavaScript API

To make it possible to create useful Fireworks extensions and customized Fireworks menus, Fireworks supports the JavaScript functions listed in this chapter. Almost any task that the user can accomplish in Fireworks with the menus, tools, or floating panels can now be done with JavaScript.

Using Fireworks API functions

There are three categories of API functions described in this chapter: Document functions, History panel functions, and Fireworks functions. The following rules apply to all functions.

Zero-based indices

All functions that take an *index* argument are zero-based members of a one-dimensional array. That is, a value of 0 represents the first item in the array, 1 represents the second item, and so on. For example, the following command deletes the third layer of the active Fireworks document:

```
fw.getDocumentDOM().deleteLayer[2];
```

Specifying the current frame or layer

All functions that take a *frameIndex* argument may be passed –1 to indicate the current frame. Similarly, all functions that take a *layerIndex* argument may be passed –1 to indicate the current layer.

Note: This is not true for symbol indices. For example, the command `fw.getDocumentDOM().duplicateSymbol(–1)` duplicates all selected symbols in the Library.

Passing NULL values

In general, passing NULL values causes an exception to be thrown. There are a few functions where it is allowable to pass NULL where a string value is expected, and these are noted in the description of those functions.

Operating on a selection

Many of the API functions in this chapter refer to a "selection" or to "selected items." These terms refer to any Fireworks objects, such as text boxes or images, that are currently selected. In most cases, the functions work even if only one item is selected. If a function requires more than one selected item, this is noted in the description of the function.

historyPalette or History panel

Several API functions reference the history palette (see page 123). Throughout the Fireworks documentation and online help, the term "palette" is reserved for discussions of a color palette, and the term "panel" is used to refer to the floating windows available within Fireworks. Therefore, although the function name contains "palette," the descriptions refer to a "panel."

Performing document functions

All of the functions listed in "Document functions" on page 59 are methods of the object that represents a Fireworks document. To perform a function on a Document object, you must first get the Document Object Model, or DOM, of the document. You then call the functions as methods of that DOM. The DOM is discussed in detail in "The Document Object Model" on page 9.

Note: You can use methods that operate on a document's DOM only on open documents.

Accessing a document's DOM

To use a DOM function with the active document, use fw.getDocumentDOM().functionName(), described below. To use a DOM function with a document other than the active document, use the syntax illustrated in "How to use the DOM" on page 10, namely:

fw.documents[*documentIndex*].functionName();

fw.getDocumentDOM()

Description Returns the Document object (see page 15) for the active document. After the object is returned, you can edit its properties to make changes to the document.

Arguments None.

Returns A Document object that represents the DOM of the active document. If there is no active document, returns NULL.

Syntax convention for document functions

Methods that operate on a document's DOM are listed in this section as dom.functionName(). However, you cannot simply type dom.functionName(). In place of dom, you must type fw.getDocumentDOM() or fw.documents[*documentIndex*]. For example:

- how a function looks in this manual: dom.addNewHotspot()
- how you must type it:

  fw.getDocumentDOM().addNewHotspot(); && operates on active document

  or

  fw.documents[*documentIndex*].addNewHotspot(); && operates on specified document

Document functions

dom.addBehavior()

Description Adds a specified behavior to the selected hotspots and slices.

Arguments *action, event, eventIndex*

- The first argument is a string that specifies the behavior to be added, such as "MM_swapImageRestore()". For a list of all behaviors that can be added, see "Using the addBehavior() function" on page 136.
- The second argument specifies the event that triggers the behavior. Acceptable values are "onMouseOver", "onMouseOut", "onLoad", and "onClick".
- The last argument is a zero-based integer that specifies the location where the behavior should be added. To specify the end location, pass –1 here.

Returns Nothing.

Example This command adds a simple rollover behavior at the end of the selected slice or hotspot.

```
fw.getDocumentDOM().addBehavior("MM_simpleRollover()", "onMouseOver", -1);
```

Related Functions dom.removeBehavior()

dom.addFrames()

Description Adds one or more frames to the document.

Arguments *howMany, where*

- The first argument is an integer that specifies how many frames to add.
- The second argument specifies where to add the frames. Acceptable values for *where* are "beginning", "before current", "after current", and "end".

Returns Nothing.

Example This command adds one frame after the current frame.

```
fw.getDocumentDOM().addFrames(1, "after current");
```

dom.addGuide()

Description Adds a guide to the document. If a guide already exists at the specified position, this function has no effect.

Arguments *position, guidekind*

- The first argument is a float value that specifies the x or y coordinate at which to add the guide.
- Acceptable values for *guidekind* are "horizontal" and "vertical". If *guidekind* is "horizontal", it is assumed that *position* is a y coordinate; if "vertical", an x coordinate.

Returns Nothing.

Example This command adds a vertical guide at the x coordinate of 217.

```
fw.getDocu mentDOM().addGuide(217, "vertical");
```

dom.addNewHotspot()

Description Adds a new hotspot fitting into the specified bounding rectangle.

Arguments *hotspot-kind, hotspot-shape, boundingRectangle*

- Acceptable values for the first argument are "hotspot" and "slice".
- Acceptable values for the second argument are "rectangle" and "oval".
- The third argument is a rectangle (see page 7) that specifies the bounds within which the hotspot is placed.

Returns Nothing.

Example This command adds a new rectangle slice with the specified coordinates.

```
fw.getDocumentDOM().addNewHotspot("slice","rectangle",{left:0, top:0, right:50,
bottom:100});
```

dom.addNewImage()

Description Adds a new, empty (transparent) image to the document.

Arguments *boundingRectangle, enterPaintModeTF*

- The first argument is a rectangle (see page 7) that specifies the bounds of the image to be added. You cannot create an image larger than the document; therefore, if you pass in a rectangle with bounds larger than the document size, you can create an image constrained to the document size.
- If the second argument is true, the application immediately enters image edit mode for the new image.

Returns Nothing.

Example This command adds an empty image that is 500 by 500 pixels in size, and then enters image edit mode.

```
fw.getDocumentDOM().addNewImage({left:0, top:0, right:500, bottom:500}, true);
```

dom.addNewLayer()

Description Adds a new layer to the document and makes it the current layer.

Arguments *name, shared*

- The first argument is a string that specifies the name for the new layer. If *name* is NULL, a new layer name is generated.
- The second argument is a boolean value that specifies whether the new layer is shared.

Returns A string value containing the name of the new layer.

Example This command adds a new, unshared layer with a default name generated by Fireworks.

```
fw.getDocumentDOM().addNewLayer(null, false);
```

dom.addNewLine()

Description Adds a new path between two points. The new path uses the document's current default path attributes and is added on the current frame and layer.

Arguments *startPoint, endPoint*

The arguments are points (see page 7) that specify the x,y coordinates between which the path is added.

Returns Nothing.

Example This command adds a new line between the specified coordinates.

```
fw.getDocumentDOM().addNewLine({x:64.5, y:279.5}, {x:393.5, y:421.5});
```

dom.addNewOval()

Description Adds a new oval fitting into the specified bounding rectangle. The oval uses the document's current default path attributes and is added on the current frame and layer.

Arguments *boundingRectangle*

The argument is a rectangle (see page 7) that specifies the bounds of the oval to be added.

Returns Nothing.

Example This command adds a new oval within the specified coordinates.

```
fw.getDocumentDOM().addNewOval({left:72, top:79, right:236, bottom:228});
```

dom.addNewRectangle()

Description Adds a new rectangle or rounded rectangle fitting into the specified bounds. The rectangle uses the document's current default path attributes and is added on the current frame and layer.

Arguments *boundingRectangle, roundness*

- The first argument is a rectangle (see page 7) that specifies the bounds within which the new rectangle is added.
- The second argument is a float value between 0 and 1 that specifies the "roundness" to use for the corners (0 is no roundness, 1 is 100% roundness).

Returns Nothing.

Example This command adds a new rectangle with no round corners within the specified coordinates.

```
fw.getDocumentDOM().addNewRectangle({left:0, top:0, right:100, bottom:100}, 0);
```

dom.addNewSinglePointPath()

Description Adds a new path consisting of a single bezier point. The path uses the default fill, stroke, and so on, and is added on the current frame and layer. The point is selected after it is added.

Arguments *controlPointFirst, mainPoint, controlPointLast, copyAttrsTF*

- The first three arguments are points (see page 7) that specify the x,y coordinates of the preceding control point, the main point, and the following control point of the bezier path.
- If *copyAttrsTF* is false, the path's stroke and fill are copied directly from the document's current stroke and fill settings. If it is true, then the path's fill is set to "none", and the brush is set to something other than "none".

Returns Nothing.

Example This command adds a new path consisting of a single bezier point at the specified coordinates, and copies the path's stroke and fill from the document's current stroke and fill settings.

```
fw.getDocumentDOM().addNewSinglePointPath({x:150, y:63}, {x:150, y:63},
{x:150, y:63}, true);
```

dom.addNewStar()

Description Adds a new star- or polygon-shaped path.

Arguments *numSides, spikiness, isStarTF, centerPoint, outsidePoint*

- The first argument is an integer that specifies how many sides the new path has.
- The second argument is a float value that controls the regularity of the star or polygon. Pass –1 to have Fireworks calculate a good value, or pass a value between 0 and 1 for manual control.
- If the third argument is true, a star with the specified number of points is created. If it is false, a regular polygon with the specified number of sides is created.
- The fourth argument specifies the center point (see page 7) of the star or polygon.
- The fifth argument specifies a point on the radius of the star or polygon.

Returns Nothing.

Example This command adds a five-sided star.

```
fw.getDocumentDOM().addNewStar(5, -1, true, {x:186, y:72}, {x:265, y:89});
```

dom.addNewSymbol()

Description Adds a new symbol to the Library and opens the symbol document for editing. Optionally adds an instance of the symbol to the document.

Arguments *type, name, addToDocTF*

- Acceptable values for the first argument are "graphic" and "button".
- The second argument is a string that specifies the name of the symbol.
- If *addToDocTF* is true, then an instance of the symbol is inserted into the center of the document. If false, the symbol is created in the document's Library, but no instance of the symbol is inserted into the document.

Returns Nothing.

Example This command adds a new graphic symbol called text to the Library, and places an instance of it in the document.

```
fw.getDocumentDOM().addNewSymbol("graphic", "text", true);
```

dom.addNewText()

Description Adds a new, empty text block within the specified bounding rectangle. (To place text in the box, use dom.setTextRuns().)

Arguments *boundingRectangle, initFromPrefsTF*

- The first argument is a rectangle (see page 7) that specifies the bounds within which to place the new text box.
- If the second argument is false, the default values for all style properties are used. If it is true, the most recent values set by the user are used.

Returns Nothing.

Example This command adds a text box with the style properties most recently used.

```
fw.getDocumentDOM().addNewText({left:43, top:220, right:102, bottom:232}, true);
```

dom.addSwapImageBehaviorFromPoint()

Description If a single hotspot or slice is selected, this function adds to it a swap-image behavior from the hotspot or slice located at *where* in the document.

Arguments *where*

The argument is a point (see page 7) that specifies the x,y coordinates of the hotspot or slice containing the swap-image behavior to be added.

Returns true if the swap-image behavior was added, false if no suitable hotspot was at the specified location.

dom.adjustExportToSize()

Description Adjusts the export settings as specified.

Arguments *sizeInBytes, okToIncreaseTF*

The first argument is a long variable that specifies the size to be used for exporting. It is used as follows.

- If a document has no slices, dom.adjustExportToSize() adjusts the export settings for the current frame so that the image is less than or equal to sizeInBytes.
- If a document has slices, dom.adjustExportToSize() adjusts the size of all exported images so that the sum of the sizes is <= sizeInBytes.

The second argument specifies whether the export file size can be increased.

- If *okToIncreaseSize* is true, and the current size is less than *sizeInBytes*, dom.adjustExportToSize() increases the quality of the export settings as much as possible, making the export size larger if necessary.
- If *okToIncrease* is false, dom.adjustExportToSize() increases the quality of the export settings as much as possible without increasing the export size.

dom.align()

Description Aligns the selection.

Arguments *alignmode*

Acceptable values for *alignmode* are "left", "right", "top", "bottom", "center vertical", and "center horizontal".

Returns Nothing.

dom.appendPointToHotspot()

Description Appends a point to the selected unclosed polygon hotspot. If an unclosed polygon hotspot is not selected, then a new polygon hotspot is created with a single point (the one passed in).

Arguments *pt, tolerance*

- The first argument is a point (see page 7) that specifies the x,y coordinates of the point to be added.
- The second argument is a float value >= 0 that specifies the tolerance between the new point and the starting point of the polyline path. If the new point is within *tolerance* of the starting point, the polyline path is closed.

Returns Nothing.

dom.appendPointToPath()

Description Appends a Bezier point to the selected path.

Arguments *contourIndex, ptToInsertBefore, controlPointFirst, mainPoint, controlPointLast*

- The first argument is a zero-based integer that specifies the contour to which the Bezier point is appended. For paths with multiple contours, the contours are in an arbitrary order.
- The second argument is a zero-based integer that specifies where on the path the new point should be placed. The new point is appended before (in front of) the point represented by this integer. To add a point to the beginning of the path, pass 0 here; to add a point to the end of the path, pass a very large number here.
- The last three arguments are points (see page 7) that specify the x,y coordinates of the preceding control point, the main point, and the following control point of the new point.

Returns Nothing.

Related Functions dom.insertPointInPath()

dom.appendPointToSlice()

Description Appends a point to the selected unclosed polygon slice. If an unclosed polygon slice is not selected, then a new polygon slice is created with a single point (the one passed in).

Arguments *pt, tolerance*

- The first argument is a point (see page 7) that specifies the x,y location of the point to be added.
- The second argument is a float value >= 0 that specifies the tolerance between the new point and the starting point of the polyline path. If the new point is within *tolerance* of the starting point, the polyline path is closed.

Returns Nothing.

dom.applyCharacterMarkup()

Description Applies the specified character markup to the selected text.

Arguments *tag*

Acceptable values for *tag* are "b", "i", and "u", for bold, italic, and underline.

Returns Nothing.

dom.applyCurrentFill()

Description Applies the document's current fill to the selection.

Arguments *noNullFillsTF*

If *noNullFillsTF* is true and the current fill is "none", then a default fill is applied instead of no fill.

Returns Nothing.

Example This command applies the current fill to the selection.

```
fw.getDocumentDOM().applyCurrentFill(true);
```

dom.applyEffects()

Description Applies the specified effects to the selection.

Arguments *effectList*

- The argument is an EffectList object (see page 33).
- If *effectList* is NULL, this function removes all effects from the selection.

Returns Nothing.

Example This command applies a drop shadow (see page 30) with an angle of 315, a blur of 4, a color of black, and a distance of 7.

```
fw.getDocumentDOM().applyEffects({ category:"Untitled", effects:[ { EffectIsVisible:true,
EffectMoaID:"{a7944db8-6ce2-11d1-8c76000502701850}", ShadowAngle:315,
ShadowBlur:4, ShadowColor:"#000000a6", ShadowDistance:7, ShadowType:0,
category:"Shadow and Glow", name:"Drop Shadow" } ], name:"Untitled" });
```

dom.applyFontMarkup()

Description Applies the specified font markup attribute to the selected text.

Arguments *fontAttribute, value*

- Acceptable values for *fontAttribute* are "size" and "face".
- If *fontAttribute* is "size", *value* must be of the form "XXXpt" to specify a point size; a simple numeric value is not allowed.

Returns Nothing.

dom.applyStyle()

Description Applies the specified style to the selection.

Arguments *styleName, styleIndex*

- The first argument is a string that specifies the style name to be applied.
- *styleIndex* is generally always zero. However, if there are multiple styles with the same name, *styleIndex* is used to resolve the tie (0 references the first style with that name, 1 references the second, and so on).

Returns Nothing.

Example This command applies the first style Fireworks encounters named "Style 7", which in this case is a default style.

```
fw.getDocumentDOM().applyStyle("Style 7", 0);
```

dom.arrange()

Description Arranges the selection.

Arguments *arrangemode*

Acceptable values for *arrangemode* are "back", "backward", "forward", and "front".

Returns Nothing.

Example This command brings the selected items to the front.

```
fw.getDocumentDOM().arrange("front");
```

dom.attachTextToPath()

Description Attaches the selected text to the selected path. If there is no text and path selected, no action is performed.

Arguments None.

Returns Nothing.

Example When there are two items selected, one a text block and the other a shape, this command attaches the text block to the shape's path.

```
fw.getDocumentDOM().attachTextToPath();
```

dom.changeGuide()

Description Moves a guide's position to a new location.

Arguments *currentPosition, newPosition, guidekind*

- The first argument is a float value that specifies the current position of the guide to be moved.
- The second argument is a float value that specifies the new position of the guide.
- Acceptable values for *guidekind* are "horizontal" and "vertical". If *guidekind* is "horizontal", it is assumed that the specified positions are y coordinates; if "vertical", x coordinates.

Returns Nothing.

Example This command moves a vertical guide from position 135 to position 275.

```
fw.getDocumentDOM().changeGuide(135, 275, "vertical");
```

dom.clipCopy()

Description Copies the selection to the Clipboard.

Arguments None.

Returns Nothing.

Example This command copies the selected items to the clipboard.

```
fw.getDocumentDOM().clipCopy();
```

dom.clipCut()

Description Cuts the selection to the Clipboard.

Arguments None.

Returns Nothing.

Example This command cuts the selected items and places them on the Clipboard.

```
fw.getDocumentDOM().clipCut();
```

dom.clipPaste()

Description Pastes the Clipboard into the document.

Arguments {*whatIfResolutionDifferent*}

- The argument is an optional string that specifies how resampling should be done if the resolution of the Clipboard doesn't match the resolution of the document. Acceptable values for *whatIfResolutionDifferent* are "resample", "do not resample", and "ask user" (displays a dialog box to let the user decide).
- If *whatIfResolutionDifferent* is omitted or NULL, "ask user" is assumed.

Returns Nothing.

Example This command pastes the Clipboard contents into the document. If there is a need for resampling, Fireworks asks the user to decide how to resample.

```
fw.getDocumentDOM().clipPaste();
```

dom.clipPasteAttributes()

Description Pastes the attributes from the Clipboard onto the selection.

Arguments None.

Returns Nothing.

Example This command applies the attributes that have been copied to the Clipboard onto the selected items.

```
fw.getDocumentDOM().clipPasteAttributes();
```

dom.clipPasteInside()

Description Pastes the Clipboard contents inside the selection.

Arguments {*whatIfResolutionDifferent*}

- The argument is an optional string that specifies how resampling should be done if the resolution of the Clipboard doesn't match the resolution of the document. Acceptable values for *whatIfResolutionDifferent* are "resample", "do not resample", and "ask user" (displays a dialog box to let the user decide).
- If *whatIfResolutionDifferent* is omitted or NULL, "ask user" is assumed.

Returns Nothing.

Example This command pastes the Clipboard contents inside the selected items. If the resolution of the Clipboard doesn't match the resolution of the document, Fireworks resamples the Clipboard contents to match the document.

```
fw.getDocumentDOM().clipPasteInside("resample");
```

dom.cloneSelection()

Description Makes an exact duplicate of the selection, placing the duplicates directly on top of the originals.

Arguments None.

Returns Nothing.

Example Copies the selected items on top of the originals.

```
fw.getDocumentDOM().cloneSelection();
```

Related Functions dom.duplicateSelection()

dom.close()

Description Closes the document.

Arguments *promptToSaveChangesTF*

If *promptToSaveChangesTF* is true, and the document has been changed since the last time it was saved, the user is prompted to save any changes to the document. If *promptToSaveChangesTF* is false, the user is not prompted, and any changes to the document are discarded.

dom.convertToPaths()

Description Converts the selected text items into editable paths.

Arguments None.

Returns Nothing.

Example This command converts the selected text items into editable paths.

```
fw.getDocumentDOM().convertToPaths();
```

dom.convertToSymbol()

Description Converts the selected item(s) to a new symbol.

Arguments *type, name*

- Acceptable values for *type* are "graphic" and "button".
- The second argument specifies a name for the new symbol.

Returns Nothing.

Example This command creates a graphic symbol out of the selected item and names it “star”.

```
fw.getDocumentDOM().convertToSymbol("graphic", "star");
```

dom.copyToHotspot()

Description Creates one or more hotspots from the selection.

Arguments *hotspotType, {whatIfMultipleSelected}*

- Acceptable values for *hotspotType* are "hotspot" and "slice".
- The second argument is an optional string that specifies how to create hotspots if multiple items are selected. Acceptable values for *whatIfMultipleSelected* are "single" (creates a single hotspot having the same bounding rectangle as the selection), "multiple" (creates one hotspot for each item), and "ask user" (displays a dialog box to let the user decide).
- If *whatIfMultipleSelected* is omitted or NULL, "ask user" is assumed.

Returns Nothing.

Example This command adds a hotspot to the selected item. If there is more than one item selected, Fireworks creates one hotspot for each item.

```
fw.getDocumentDOM().copyToHotspot("hotspot", "multiple");
```

dom.cropSelection()

Description Crops the selection to the specified rectangle.

Arguments *boundingRectangle*

The argument is a rectangle (see page 7) that specifies the bounds within which the selection should be cropped.

Returns Nothing.

dom.deleteFrames()

Description Deletes one or more frames.

Arguments *frameIndex, howMany*

- The first argument is a zero-based integer that specifies the location at which to begin deleting frames. To specify the current frame, pass –1.
- The second argument specifies how many frames to delete.

Returns Nothing.

dom.deleteLayer()

Description Deletes a layer.

Arguments *layerIndex*

The argument is a zero-based integer that specifies the layer to be deleted. To specify the current layer, pass –1.

Returns Nothing.

Example This command deletes the current layer.

```
fw.getDocumentDOM().deleteLayer(-1);
```

dom.deleteSelection()

Description Deletes the selection, or the pixel-selection if Fireworks is in image edit mode.

Arguments *fillDeletedAreaTF*

- The argument is ignored if you are not in image edit mode.
- If Fireworks is in image edit mode and *fillDeletedAreaTF* is true, the deleted pixels are filled with the current fill color. If false, the deleted pixels are filled to transparent.

Returns Nothing.

Example If Fireworks is not in image edit mode, this command deletes the selected items. If Fireworks is in image edit mode, this command fills the selected items to transparent.

```
fw.getDocumentDOM().deleteSelection(false);
```

dom.deleteSymbol()

Description Deletes the specified symbols from the Library.

Arguments *symbolName*

The argument is the name of the symbol to be deleted from the Library. If more than one symbol exists with this name, only the first symbol is deleted.

- To delete all selected symbols from the Library (not document), pass NULL.
- If the deleted symbols contain any active instances in the document, the instances are also deleted.

Returns Nothing.

Example This command deletes all of the selected symbols from the Library, and also deletes any active instances from the document.

```
fw.getDocumentDOM().deleteSymbol(null);
```

dom.detachInstanceFromSymbol()

Description Breaks the links between the selected instances and the owning symbols.

Arguments None.

Returns Nothing.

dom.detachTextFromPath()

Description Splits the selected text-on-a-path into its original text and path items.

Arguments None.

Returns Nothing.

dom.distribute()

Description Distributes the selection along a vertical or horizontal dimension.

Arguments *dimension*

Acceptable values for *dimension* are "vertical" and "horizontal".

Returns Nothing.

dom.distributeLayerToFrames()

Description Distributes the items on the specified layer to the frames of the document, adding frames if necessary. The first item on the layer goes to the first frame, the second item to the second frame, and so on. New frames are added to the document, if necessary. If there is only one item in the specified layer, this function has no effect.

Arguments *layerIndex*

The argument is a zero-based integer that specifies the layer containing the items to be distributed. To specify the current layer, pass –1.

Returns Nothing.

dom.distributeSelectionToFrames()

Description Distributes the selected items to the frames of the document, adding frames if necessary. The first item goes to the current frame, the second item to the next frame, and so on. If there is only one item selected, this function has no effect.

Arguments None.

Returns Nothing.

dom.duplicateFrame()

Description Duplicates a frame.

Arguments *frameIndex, howMany, where, dupeSelectionOnlyTF*

- The first argument is a zero-based integer that specifies the frame to duplicate. To specify the current frame, pass –1.
- The second argument is an integer that specifies how many copies of the frame to make.
- Acceptable values for *where* are "beginning", "before current", "after current", and "end".
- If *dupeSelectionOnlyTF* is true, then only items in the specified frame that are selected are duplicated to the new frame.

Returns Nothing.

Example This command makes one copy of the current frame, and places the new frame after the current frame.

```
fw.getDocumentDOM().duplicateFrame(-1, 1, "after current", false);
```

dom.duplicateLayer()

Description Duplicates a layer.

Arguments *layerIndex*

The argument is a zero-based integer that specifies the layer to duplicate. To specify the current layer, pass –1.

Returns Nothing.

Example This command duplicates the current layer.

```
fw.getDocumentDOM().duplicateLayer(-1);
```

dom.duplicateSelection()

Description Makes an exact duplicate of the selection, offsetting it slightly from the original.

Arguments None.

Returns Nothing.

Example This command duplicates the selected items.

```
fw.getDocumentDOM().duplicateSelection();
```

Related Functions dom.cloneSelection()

dom.duplicateSelectionToFrameRange()

Description Duplicates the selection to a range of frames of the document.

Arguments *frameIndexFirst, frameIndexLast*

The arguments are zero-based integers that specify the range of frames (inclusive) to which the items should be copied. To specify the current frame, pass –1.

- If both arguments are the same, copies are placed only on that frame.
- If the range includes the current frame, copies are not placed on that frame.

Returns Nothing.

dom.duplicateSelectionToFrames()

Description Duplicates the selection to specified frames of the document.

Arguments *whichFrames*

- Acceptable values for *whichFrames* are "all", "previous", "next", and "end".
- Note that "end" means the last frame of the document; it does not add a new frame.

Returns Nothing.

dom.duplicateSymbol()

Description Duplicates the specified symbol.

Arguments *symbolIndex*

The argument is a zero-based integer that specifies the symbol to be duplicated.

- To duplicate all selected symbols in the Library (not document), pass a value of –1.
- Duplicating a linked symbol results in a non-linked duplicate.

Returns Nothing.

dom.duplicateSymbolForAlias()

Description If any symbol instances are selected, this function makes duplicate symbols of all symbols pointed to by those instances. The selected instances are updated to point to the new duplicate copy of the symbols. Duplicate symbols always result in non-linked duplicates. (The use of the word “alias” in the function name corresponds to an “instance” in a Fireworks document.)

Arguments None.

Returns Nothing.

dom.enterPaintMode()

Description Enters image edit mode on the selected items. Has no effect if nothing is selected or if a non-image item is selected.

Arguments None.

Returns Nothing.

dom.exitPaintMode()

Description Leaves image edit mode. Has no effect if Fireworks is not in image edit mode.

Arguments None.

Returns Nothing.

dom.exportOptions.loadColorPalette()

Description Replaces the values in dom.exportOptions.paletteEntries with those in the specified GIF or ACT file. This function also sets dom.exportOptions.paletteMode to "custom". For more information, see "ExportOptions" on page 35.

Arguments *fileURL*

The argument is a fileURL that specifies the GIF or ACT file to be used to replace the color palette.

Returns true if the file was read successfully, false if the file was not of the expected format or was not read successfully for any other reason.

dom.exportOptions.saveColorPalette()

Description Saves the values in dom.exportOptions.paletteEntries to the specified color palette (ACT file). This function does not modify the document. For more information, see "ExportOptions" on page 35.

Arguments *fileURL*

The argument is a fileURL that specifies the name of the file to which the color palette should be saved. Do not specify a file extension; the .act extension is added automatically.

Returns Nothing.

dom.exportTo()

Description Exports the document as specified.

Arguments *fileURL, {exportOptions}*

- The first argument is a fileURL that specifies the name of the exported file.
- The second argument is optional, and is an ExportOptions object (see page 35). If this argument is omitted or NULL, the document's current export options settings are used. If values are passed in with *exportOptions*, they are used for this export operation only; they do not changed the document's exportFormatOptions property.

Returns true if the file was successfully exported, false otherwise.

dom.fillSelectedPixels()

Description When the selection is an image and Fireworks is in image edit mode, fills the selected pixels with the current fill or generates a new pixel selection.

Arguments *clickPt, p1, p2, p3, fillSelectionOnlyTF, tolerance, edgemode, featherAmt*

- The first argument is a point (see page 7) that specifies the x,y coordinates of the pixel to be filled or generated.
- The second, third, and fourth arguments are points that specify the fill-vector. These arguments are ignored if the current fill does not use a fill-vector.
- If the fifth argument (*fillSelectionOnlyTF*) is true, the remaining arguments are ignored. If it is false, then the current pixel selection is ignored, and a new one is generated using the values passed for *tolerance*, *edgemode*, and *featherAmt*. (This behavior is the same as if the magic wand tool were used at location *clickPt*.)
- The sixth argument (*tolerance*) is an integer between 0 and 255 inclusive that specifies the tolerance for selecting pixels.
- Acceptable values for *edgemode* are "hard edge", "antialias", and "feather".
- The last argument (*featherAmt*) is an integer between 0 and 32000 inclusive that specifies the number of pixels to feather. This value is ignored if *edgemode* is not "feather".

Returns Nothing.

Example This command fills the selection with a hard edge, tolerance set to 32.

```
fw.getDocumentDOM().fillSelectedPixels({x:207, y:199}, {x:207, y:199}, {x:207, y:199},
{x:207, y:199}, false, 32, "hard edge", 0);
```

dom.filterSelection()

Description Applies the specified pixel filter to the selection. Non-image items are converted into images before the filter is applied. Only external filters that are capable of also being Live Effects can be applied using this function. To apply other types of external filters, use dom.filterSelectionByName().

Arguments *LiveEffect*

The argument is an Effect object (see page 27).

Returns Nothing.

Example This command runs the selected pixels through the hue/saturation filter, and then sets hue to 30 and saturation to 20.

```
fw.getDocumentDOM().filterSelection({
    EffectMoaID:"{3439b08d-1922-11d3-9bde00e02910d580}",
    hls_colorize:true, hue_amount:30, lightness_amount:0, saturation_amount:20
});
```

dom.filterSelectionByName()

Description Applies the specified pixel filter to the selection. This is applied as a permanent (but undoable) action, not as a Live Effect. (To apply filters that can also be Live Effects, you can use dom.filterSelection().) This function always displays a dialog box.

Arguments *category, name*

- The first argument is a string that specifies the category of the pixel filter to be applied. Acceptable values depend on which filters you have installed.
- The second argument is a string that specifies the name of the pixel filter to be applied. Acceptable values depend on which filters you have installed.

Returns Nothing.

dom.findExportFormatOptionsByName()

Description Looks for a set of Export Settings that have been saved with the specified name.

Arguments *name*

The argument is a string that specifies the name of the set of Export Settings to look for.

Returns If there is a set of Export Settings with the specified name, returns an object representing it; otherwise returns NULL.

dom.flattenDocument()

Description Flattens the entire document into a single pixel image. This is the same behavior as the Merge Layers command.

Arguments None.

Returns Nothing.

dom.flattenSelection()

Description Flattens the selection into a single pixel image. This is the same behavior as the Merge Images command.

Arguments None.

Returns Nothing.

dom.getFontMarkup()

Description Gets a font markup attribute for the selected text.

Arguments *fontAttribute*

Acceptable values for *fontAttribute* are "size", "color", and "face".

Returns A string specifying the markup value. Returns NULL if the text has multiple attributes or if the selection contains no text.

dom.getPixelMask()

Description Gets the current pixel-selection mask.

Arguments None.

Returns The mask for the current pixel selection. Returns NULL if Fireworks is not in image edit mode, or if there is no pixel selection. For information on the format of mask variables, see “Mask” on page 6.

dom.getSelectionBounds()

Description Gets the bounding rectangle of the selection.

Arguments None.

Returns A rectangle (see page 7). Returns NULL if nothing is selected.

dom.getShowGrid()

Description Determines if the grid is visible.

Arguments None.

Returns true if the grid is visible, false if it is not.

dom.getShowRulers()

Description Determines if the rulers are visible.

Arguments None.

Returns true if the rulers are visible, false if they are not.

dom.getSnapToGrid()

Description Determines if the Snap to Grid function is active.

Arguments None.

Returns true if the Snap to Grid function is active, false if it is not.

dom.getTextAlignment()

Description Gets the alignment of selected text.

Arguments None.

Returns One of the following strings: "left", "center", "right", "justify", "stretch", "vertical left", "vertical center", "vertical right", "vertical justify", or "vertical stretch". Returns NULL if the text has multiple alignments or if the selection contains no text.

dom.group()

Description Groups the selection. To ungroup items, use dom.ungroup().

Arguments *type*

The argument specifies how to group the items. Acceptable values are "normal", "mask to image", and "mask to path".

Returns Nothing.

Example This command sets the selected group to mask to the image.

```
fw.getDocumentDOM().group("mask to image");
```

dom.hasCharacterMarkup()

Description Determines if the selected text has the specified character markup.

Arguments *tag*

Acceptable values for *tag* are "b", "i", and "u", for bold, italic, and underline.

Returns true if the text has the specified character markup, false if it does not or if only part of the text has the markup.

dom.hideSelection()

Description Hides the selection. To redisplay it, use dom.showAllHidden().

Arguments None.

Returns Nothing.

dom.importFile()

Description Imports the specified file at the specified location.

Arguments *fileURL, boundingRectangle, maintainAspectRatioTF*

- The first argument is the fileURL of the file to be imported.
- The second argument is a rectangle (see page 7) that specifies the size to make the imported file. If *boundingRectangle* is specified with left == right and top == bottom, the file is brought in unscaled with its top-left corner at that point, and the third argument is ignored.
- If the third argument (*maintainAspectRatioTF*) is true, the file is scaled to the largest size that fits within *boundingRectangle* while retaining the file's current aspect ratio. (This is a handy option for creating thumbnails.) If it is false, the file is scaled to fill *boundingRectangle*.

Returns Nothing.

Example This command imports the specified file and maintains its aspect ratio.

```
fw.getDocumentDOM().importFile("file:///C|/images", {left:25, top:50, right:100,
bottom:250}, true);dom.importSymbol()
```

Description Imports the specified external graphics file (for example, GIF, JPEG, Fireworks document) into the Library of the document.

Arguments *fileURL, addToDocTF, allowUI-TF*

- The first argument is the fileURL of the file to be imported into the Library.
- If the second argument (*addToDocTF*) is true, then the symbol is added to the Library and an instance of the symbol is inserted into the center of the document. If it is false, the symbol is added only to the Library.
- If the third argument (*allowUI-TF*) is true, and *fileURL* is a Fireworks document containing symbols, then a dialog box allows the user to specify which symbols to import from the external file. If it is false, then all symbols in the external file are imported.

Returns Nothing.

dom.insertPointInPath()

Description Inserts a Bezier point in the selected path. This function is similar to dom.appendPointToPath(), but includes a t-parameter argument which gives you finer control over where the point is inserted.

Arguments *contourIndex, ptToInsertBefore, tParameter, controlPointFirst, mainPoint, controlPointLast*

- The first argument is a zero-based integer that specifies the contour into which the Bezier point is inserted. For paths with multiple contours, the contours are in an arbitrary order.
- The second argument is a zero-based integer that specifies where on the path the new point should be placed. The new point is appended before (in front of) the point represented by this integer: to add a point to the beginning of the path, pass 0 here; to add a point to the end of the path, pass a very large number here.
- The third argument is a float value between 0 and 1 that specifies where in the Bezier segment to insert the new point.
- The last three arguments are points (see page 7) that specify the x,y coordinates of the preceding control point, the main point, and the following control point of the new point.

Returns Nothing.

Related Functions dom.appendPointToPath()

dom.joinPaths()

Description Joins the selected paths.

Arguments None.

Returns Nothing.

dom.knifeElementsFromPoint()

Description When the user clicks a single point while using the knife tool, this function knifes additional items within the specified tolerance. This is similar to using the knife tool with a single click.

Arguments *from, tolerance*

- The first argument is a point (see page 7) that specifies the x,y coordinates of the point that the user clicked.
- The second argument is a float value >= 0 that specifies the tolerance within which items are knifed.

Returns true if anything was knifed, false otherwise.

Related Functions dom.knifeElementsFromPoints()

dom.knifeElementsFromPoints()

Description When the user drags while using the knife tool, this function knifes additional items within the specified tolerance. This is similar to using the knife tool with a drag operation.

Arguments *from, to, tolerance*

- The first argument is a point (see page 7) that specifies the x,y coordinates of the point where the user clicked and started to drag.
- The second argument is a point that specifies the x,y coordinates of the point where the user ended the drag operation.
- The last argument is a float value >= 0 that specifies the tolerance within which items are knifed.

Returns true if anything was knifed, false otherwise.

Related Functions dom.knifeElemenstFromPoint()

dom.makeFind()

Description Creates an object of class Find, to do a find-and-replace operation in this document.

Arguments *findSpec*

The argument is a Find object (see page 20).

dom.makeGoodNativeFilePath()

Description Ensures that the specified fileURL ends in a .png extension. Does not affect name of the file on disk.

Arguments *fileURL*

The argument is the fileURL of the file whose extension should be changed to .png if necessary.

Returns A string containing the fileURL with a .png extension.

Example The following command returns "file:///My Documents/image01.png".

```
fw.getDocumentDOM().makeGoodNativeFilePath("file:///My Documents/image01.ping")
```

dom.makeActive()

Description Makes the selected document active for editing.

Arguments None.

Returns Nothing.

dom.modifyPointOnPath()

Description Modifies an existing point on the selected path.

Arguments *contourIndex, ptToModify, controlPointFirst, mainPoint, controlPointLast, reapplyAttrsTF, closePathTF*

- The first argument is a zero-based integer that specifies the contour into which the Bezier point is inserted. For paths with multiple contours, the contours are in an arbitrary order.
- The second argument is a zero-based integer that specifies the point to be modified.
- The next three arguments are points (see page 7) that specify the x,y coordinates of the preceding control point, the main point, and the following control point of the new point.
- If the sixth argument (*reapplyAttrsTF*) is true, the path has the document's current fill, stroke, and so on reapplied to it. If it is false, then the path attributes are not changed.
- If the last argument (*closePathTF*) is true, the path is marked as closed after modifying the point. If it is false, then the path retains its original open or closed value.

Returns Nothing.

dom.moveBezierHandleBy()

Description Moves the specified point's Bezier handles by a certain amount.

Arguments *whichPath, contourIndex, ptToModify, deltaControlPointFirst, deltaControlPointLast*

- The first argument is a zero-based integer that specifies an index into the list of selected items, indicating which item contains the Bezier handles to be moved.
- The second argument is a zero-based integer that specifies the contour containing the handles to be moved. For paths with multiple contours, the contours are in an arbitrary order.
- The third argument is a zero-based integer that specifies the point whose handles are moved.
- The last two arguments are points that specify the x,y coordinate values by which the preceding control point and the following control point of *ptToModify* are moved. For example, passing `{x:1,y:2}` specifies a location right by one pixel and down by two pixels.

Returns Nothing.

dom.moveFillVectorHandleBy()

Description If the selection has a fill that uses a fill-vector (for example, a gradient fill of some sort), this function adjusts the handles of the fill vector. If it does not, this function has no effect.

Arguments *delta, whichHandle, constrainTF, moveJustOneTF*

- The first argument is a point (see page 7) that specifies the x,y coordinate values by which the handle is moved. For example, passing `{x:1,y:2}` specifies a location right by one pixel and down by two pixels.
- The second argument specifies which handle to move, and can be one of the following values: `"start"`, `"end1"`, `"end2"`, `"rotate1"`, or `"rotate2"`. (Some fills ignore `"end2"`.) Use `"rotate1"`, or `"rotate2"` to rotate the `end1` or `end2` point around the start point.
- If the third argument (*constrainTF*) is `true`, movement is constrained to 45-degree increments.
- If the last argument (*moveJustOneTF*) is `true`, then only the specified handle is moved. If it is `false`, other handles may move in sync when the specified handle is moved.

Returns Nothing.

dom.moveMaskGroupContentsBy()

Description If the selection is a mask group, this function moves the contents within the mask group by the specified amount.

Arguments *delta*

The argument is a point (see page 7) that specifies the x,y coordinate values by which the contents within the mask group are moved. For example, passing ({x:1,y:2}) specifies a location right by one pixel and down by two pixels.

Returns Nothing.

dom.movePointOnHotspotBy()

Description If the selection is a hotspot or slice of the polyline variety, this function moves a point on the hotspot's path by the specified amount.

Arguments *ptToModifyIndex, delta*

- The first argument is a zero-based integer that specifies which point on the path is to be moved.
- The second argument is a point (see page 7) that specifies the x,y coordinate values by which the point is moved. For example, passing ({x:1,y:2}) specifies a location right by one pixel and down by two pixels.

Returns Nothing.

dom.moveSelectedBezierPointsBy()

Description If the selection contains at least one path with at least one Bezier point selected, this function moves all selected Bezier points on all selected paths by the specified amount.

Arguments *delta*

The argument is a point (see page 7) that specifies the x,y coordinate values by which the selected Bezier points are moved. For example, passing ({x:1,y:2}) specifies a location right by one pixel and down by two pixels.

Returns Nothing.

dom.moveSelectionBy()

Description Moves all of the selected items by the specified amount, or makes a copy of them offset from the original by the specified amount. (Unlike moveSelectedBezierPointsBy(), the selection status of Bezier points is ignored.)

Arguments *delta, makeCopyTF*

- The first argument is a point (see page 7) that specifies the x,y coordinate values by which the selection is moved. For example, passing ({x:1,y:2}) specifies a location right by one pixel and down by two pixels.
- If the second argument is true, the items are copied instead of moved.

Returns Nothing.

Example Moves the selected items 62 pixels to the right and 84 pixels down.

```
fw.getDocumentDOM().moveSelectionBy({x:62, y:84}, false, false);
```

dom.moveSelectionTo()

Description Moves or copies the selection to the specified location.

Arguments *location, makeCopyTF*

- The first argument is a point (see page 7) that specifies the x,y coordinate values of the location to which the selection is moved or copied.
- If the second argument is true, the selection is copied instead of moved.

Returns Nothing.

dom.moveSelectionToFrame()

Description Moves or copies the selection to the specified frame.

Arguments *frameIndex, makeCopyTF*

- The first argument is a zero-based integer that specifies the frame to which the selection is moved or copied. To specify the current frame, pass –1.
- If the second argument is true, the selection is copied instead of moved.

Returns Nothing.

dom.moveSelectionToLayer()

Description Moves or copies the selection to the specified layer.

Arguments *layerIndex, makeCopyTF, {whatIfMultipleSelected}*

- The first argument is a zero-based integer that specifies the layer to which the selection should be moved or copied. To specify the current layer, pass –1.
- If the second argument is true, the selection is copied instead of moved.
- The third argument (*whatIfMultipleSelected*) is an optional string that is used only if the destination is a web layer and *makeCopyTF* is true. It specifies how to create hotspots if multiple items are selected. Acceptable values for *whatIfMultipleSelected* are "single" (creates a single hotspot having the same bounding rectangle as the selection), "multiple" (creates one hotspot for each item), and "ask user" (displays a dialog box to let the user decide).
- If *whatIfMultipleSelected* is omitted or NULL, "ask user" is assumed.

Returns Nothing.

dom.moveSelectionToNewLayer()

Description Makes a new layer with a default name, then moves or copies the selection to that new layer.

Arguments *makeCopyTF*

If the argument is true, the selected items are copied instead of moved.

Returns Nothing.

dom.pathCrop()

Description Performs a Crop operation on the selected paths.

Arguments None.

Returns Nothing.

dom.pathExpand()

Description Performs an Expand operation on the selected paths.

Arguments *width, miter, cap, join*

- The first argument is a float value that specifies the new width of the selected paths, in pixels.
- The second argument is a float value that specifies the new miter angle of the selected paths, in pixels. This argument is ignored if *join* is not "miter".
- Acceptable values for *cap* are "butt", "square", and "round".
- Acceptable values for *join* are "bevel", "round", and "miter".

Returns Nothing.

dom.pathInset()

Description Performs an Inset operation on the selected paths.

Arguments *width, miter, join*

- The first argument is a float value that specifies the new width of the selected paths, in pixels.
- The second argument is a float value that specifies the new miter angle of the selected paths, in pixels. This argument is ignored if *join* is not "miter".
- Acceptable values for *join* are "bevel", "round", and "miter".

Returns Nothing.

dom.pathIntersect()

Description Performs an Intersect operation on the selected paths.

Arguments None.

Returns Nothing.

dom.pathPunch()

Description Performs a Punch operation on the selected paths.

Arguments None.

Returns Nothing.

dom.pathSimplify()

Description Performs a Simplify operation on the selected paths.

Arguments *limit*

The argument is a float value that specifies how much to simplify. This value corresponds to the value in the Modify > Alter Path > Simplify dialog box.

Returns Nothing.

dom.pathUnion()

Description Performs a Union operation on the selected paths.

Arguments None.

Returns Nothing.

dom.rebuildColorTable()

Description Rebuilds the color table for the current export settings of the document. This is the same behavior as choosing Rebuild Color Table from the Color Table panel.

Arguments None.

Returns Nothing.

dom.redo()

Description Redoes the last action that was Undone in the document.

Arguments None.

Returns Nothing.

dom.reflectSelection()

Description Reflects the selection vertically or horizontally, or both.

Arguments *horizTF, vertTF, opts*

- If the first value is true, the items are reflected horizontally.
- If the second value is true, the items are reflected vertically.
- Acceptable values for *opts* are "transformAttributes", "autoTrimImages", and "autoTrimImages transformAttributes".

Returns Nothing.

dom.removeAllGuides()

Description Removes all guides of the specified type.

Arguments *guidekind*

Acceptable values for *guidekind* are "horizontal" and "vertical".

Returns Nothing.

dom.removeBehavior()

Description Removes one or all behavior events from the selected hotspots and slices.

Arguments {*event*}, {*eventIndex*}

- Both arguments are optional; if they are omitted, this function removes all events from selected hotspots and slices.
- The first argument specifies the event that triggers the behavior. This argument is ignored by Fireworks.
- The second argument is a zero-based integer that specifies the location of the behavior to be removed. To specify the end location, pass –1 here.

Returns Nothing.

Related Functions dom.addBehavior()

dom.removeBrush()

Description Sets the brush of the selection to None.

Arguments None.

Returns Nothing.

dom.removeCharacterMarkup()

Description Reapplies the default value for the specified markup type to the text in the selection.

Arguments *tag*

Acceptable values for *tag* are "b", "i", and "u", for bold, italic, and underline.

Returns Nothing.

dom.removeFontMarkup()

Description Reapplies the default value for the specified font attribute to the text in the selection.

Arguments *fontAttribute*

Acceptable values for *fontAttribute* are "size", "color", and "face".

Returns Nothing.

dom.removeFill()

Description Sets the fill of the selection to None.

Arguments None.

Returns Nothing.

dom.removeGuide()

Description Removes the specified guide. If no guide is at that position, this function has no effect.

Arguments *position, guidekind*

- The first argument is a float value that specifies the position of the guide to be removed.
- Acceptable values for *guidekind* are "horizontal" and "vertical". If *guidekind* is "horizontal", it is assumed that *position* is a y coordinate; if "vertical", an x coordinate

Returns Nothing.

dom.removeTransformation()

Description Removes the transformations, if any, from the selected text or instances.

Arguments None.

Returns Nothing.

dom.reorderFrame()

Description Moves or copies the specified frame before another specified frame.

Arguments *frameToMove, frameToPutItBefore, makeCopyTF*

- The first argument is a zero-based integer that specifies which frame to move or copy.
- The second argument is a zero-based integer that specifies which frame you want to move or copy the frame before. That is, if you pass 1 for *frameToMove* and 0 for *frameToPutItBefore*, the second frame is placed before the first frame.
- If the third argument is true, the specified frame is copied instead of moved.

Returns Nothing.

Example This command moves the third frame before the first frame.

```
fw.getDocumentDOM().reorderFrame(2, 0, false);
```

dom.reorderLayer()

Description Moves or copies the specified layer before another specified layer.

Arguments *layerToMove, layerToPutItBefore, makeCopyTF*

- The first argument is a zero-based integer that specifies which layer to move or copy.
- The second argument is a zero-based integer that specifies which layer you want to move or copy the layer before. That is, if you pass 1 for *layerToMove* and 0 for *layerToPutItBefore*, the second layer is placed before the first layer.
- If the third argument is true, the specified layer is copied instead of moved.

Returns Nothing.

dom.replaceButtonTextStrings()

Description Replaces all text items within the document that are defined as button text items with the specified string. (Button text items are defined as the topmost text items on each frame.) Acts on both selected and unselected items.

Arguments *newString, uniformAttrs*

- The first argument specifies the string to be used as replacement text.
- If *uniformAttrs* is false, each character retains the attributes of the character formerly in its position; that is, Fireworks does its best to preserve the existing formatting. If *uniformAttrs* is true, all characters assume the attributes of the first character in the string being replaced.

Returns Nothing.

Related Functions dom.replaceButtonTextStringsInInstances()

dom.replaceButtonTextStringsInInstances()

Description Replaces selected button text items with the specified string. (Button text items are defined as the topmost text items on each frame.)

Arguments *newString, uniformAttrs*

- The first argument specifies the string to be used as replacement text.
- If *uniformAttrs* is false, each character retains the attributes of the character formerly in its position; that is, Fireworks does its best to preserve the existing formatting. If *uniformAttrs* is true, all characters assume the attributes of the first character in the string being replaced.

Returns Nothing.

Related Functions dom.replaceButtonTextStrings()

dom.replaceTextString()

Description Replaces the text of all selected text items with the specified string.

Arguments *newString, uniformAttrs*

- The first argument specifies the string to be used as replacement text.
- If *uniformAttrs* is false, each character retains the attributes of the character formerly in its position; that is, Fireworks does its best to preserve the existing formatting. If *uniformAttrs* is true, all characters assume the attributes of the first character in the string being replaced.

Returns Nothing.

dom.resizeSelection()

Description Resizes the selection to the specified pixel width and height, keeping the top left corner of the selection in place.

Arguments *width, height*

The arguments are integers that specify the new width and height in pixels.

Returns Nothing.

dom.reversePathTextDirection()

Description For all text-on-a-path items in the selection, reverses the direction of the text along the path.

Arguments None.

Returns Nothing.

dom.rotateDocument()

Description Rotates the entire document 90, 180, or 270 degrees clockwise. Note that 270 degrees is the same behavior as rotating 90 degrees counter-clockwise (CCW).

Arguments *rotationAmount*

Acceptable values for *rotationAmount* are 90, 180, and 270.

Returns Nothing.

dom.rotateSelection()

Description Rotates the selection clockwise the specified number of degrees.

Arguments *rotationDegrees, opts*

- The first argument is a float value that specifies the number of degrees to rotate the selection.
- Acceptable values for *opts* are "transformAttributes", "autoTrimImages", and "autoTrimImages transformAttributes".

Returns Nothing.

dom.save()

Description Saves the document in its default location. Upon a successful save, the document's isDirty flag is cleared.

Arguments {*okToSaveAsTF*}

If *okToSaveAsTF* is true or omitted and the file has never been saved, then the Save As dialog box is displayed. If *okToSaveAsTF* is false and the file has never been saved, the file is not saved.

Returns true if the save operation completes successfully, false otherwise.

dom.saveCopyAs()

Description Save a copy of the document in a specified directory with a specified name. This function does not affect the document's filePathForSave or isDirty properties.

Arguments *fileURL*

The argument is a fileURL specifying the directory and name under which the copy should be saved.

Returns true if the save operation completes successfully, false otherwise.

dom.scaleSelection()

Description Scales the selection in the x and y axes.

Arguments *xScaleAmount, yScaleAmount, opts*

- The first two arguments are float values that specify the amount to scale the selection in the x and y axes. Acceptable values are 0.0 or greater; a value of 1 represents 100%, 2 represents 200%, and so on.
- Acceptable values for *opts* are "transformAttributes", "autoTrimImages", and "autoTrimImages transformAttributes".

Returns Nothing.

Example This command scales the selected items to approximately two-thirds (67%) and turns on auto trim images and transform attributes.

```
fw.getDocumentDOM().scaleSelection(0.67, 0.67, "autoTrimImages transformAttributes");
```

dom.selectAdjustPixelSel()

Description Grows or shrinks the pixel selection by the specified number of pixels, selects a border of pixels, or smooths the edge of the pixel selection.

Arguments *whatToDo, amount*

Acceptable values for *whatToDo* are "expand", "contract", "border", and "smooth". Any long value is acceptable for *amount*.

- Use "expand" to grow the pixel selection outward by the number of pixels specified by *amount*.
- Use "contract" to shrink the pixel selection inwards by the number of pixels specified by *amount*.
- Use "border" to select a band of pixels the width of *amount* around the edge of the pixel selection.
- Use "smooth" to smooth out the edge of the pixel selection by *amount*.

Returns Nothing.

dom.selectAll()

Description Selects all items in the current layer and frame.

Arguments None.

Returns Nothing.

dom.selectChildren()

Description Selects the children, if any, of the selection. For example, if a group is selected, the selection changes from the group to the individual members of that group.

Arguments None.

Returns Nothing.

Related Functions dom.selectParents()

dom.selectFeather()

Description If Fireworks is in image edit mode and a pixel selection is active, this function feathers the selection by the specified number of pixels.

Arguments *featherAmount*

The argument is an integer that specifies the number of pixels by which to feather the selection.

Returns Nothing.

dom.selectInverse()

Description If Fireworks is in image edit mode and a pixel selection is active, this function inverts the pixel selection.

Arguments None.

Returns Nothing.

dom.selectNone()

Description Deselects any selected items.

Arguments None.

Returns Nothing.

dom.selectParents()

Description Selects the parents, if any, of the selection. That is, if all members of a group are selected, then the individual members are deselected and the group is selected.

Arguments None.

Returns Nothing.

Related Functions dom.selectChildren()

dom.selectSimilar()

Description If Fireworks is in image edit mode and a pixel selection is active, this function selects all pixels in the current image that are within the specified tolerance of the average color in the current pixel selection.

Arguments *tolerance, edgemode, featherAmt, combinemode*

- The first argument is an integer between 0 and 255 inclusive that specifies the tolerance for selecting pixels.
- Acceptable values for *edgemode* are "hard edge", "antialias", and "feather".
- The third argument (*featherAmt*) is an integer that specifies the number of pixels to feather. This value is ignored if *edgemode* is not "feather".
- The last argument (*combinemode*) specifies how to combine the new selection mask with the existing mask. Acceptable values are "replace", "add", "subtract", and "intersect".

Returns Nothing.

Related Functions dom.selectSimilarFromPoint()

dom.selectSimilarFromPoint()

Description Behavior is almost identical to dom.selectSimilar(), except that the new mask is calculated from the color at the specified location in the image, rather than from the average color in the selection.

Arguments *where, tolerance, edgemode, featheramount, combinemode*

- The first argument is a point (see page 7) specifies the x, y coordinates of the pixel whose color is used to calculate the new mask.
- The second argument is an integer between 0 and 255 inclusive that specifies the tolerance for selecting pixels.
- Acceptable values for *edgemode* are "hard edge", "antialias", and "feather".
- The fourth argument (*featherAmt*) is an integer that specifies the number of pixels to feather. This value is ignored if *edgemode* is not "feather".
- The last argument (*combinemode*) specifies how to combine the new selection mask with the existing mask. Acceptable values are "replace", "add", "subtract", and "intersect".

Returns Nothing.

Related Functions dom.selectSimilar()

dom.setAnimInstanceLoopCount()

Description Sets the loop count of the selected instances of multi-frame graphic symbols.

Arguments *loopCount*

The argument is an integer that corresponds to the loop count value seen in the object inspector when a multi-frame graphic instance is selected.

Returns Nothing.

dom.setAnimInstanceStartFrame()

Description Sets the start frame of the selected instances of multi-frame graphic symbols.

Arguments *startFrame*

The argument is an integer that corresponds to the start frame value seen in the object inspector when a multi-frame graphic instance is selected.

Returns Nothing.

dom.setBlendMode()

Description Specifies the blend mode of the selection.

Arguments *mode*

Acceptable values for *mode* are "normal", "multiply", "screen", "darken", "lighten", "difference", "hue", "saturation", "color", "luminosity", "invert", "tint", and "erase".

Returns Nothing.

dom.setBrush()

Description Sets the selection to the specified brush.

Arguments *brush*

The argument is a Brush object (see page 24).

Returns Nothing.

Related Functions dom.setBrushColor(), dom.setBrushName(), dom.setBrushNColorNTexture(), dom.setBrushPlacement()

dom.setBrushColor()

Description Sets the brush color of the selection to the specified color.

Arguments *color*

The argument is a color string in the format "#rrggbb" or "#rrggbbaa".

Returns Nothing.

Related Functions dom.setBrushNColorNTexture()

dom.setBrushName()

Description Renames a brush. Does not change the brush category.

Arguments *category, currentName, newName*

- The first argument is a string that specifies the category of the brush to be renamed.
- The second argument is a string that specifies the current name of the brush.
- The last argument is a string that specifies what the new name of the brush should be.

Returns Nothing.

dom.setBrushNColorNTexture()

Description Sets the selection to the specified brush, brush color, and brush texture.

Arguments *brush, color, texture-name*

- The first argument is a Brush object (see page 24).
- The second argument is a color string in the format "#rrggbb" or "#rrggbbaa".
- The last argument is the name of the texture to be applied.

Returns Nothing.

Related Functions dom.setBrushColor()

dom.setBrushPlacement()

Description Specifies the brush placement of the stroke on the selection.

Arguments *placement*

Acceptable values for *placement* are "inside", "center", and "outside".

Returns Nothing.

dom.setButtonAutoSlice()

Description If the user is editing a button document, this function turns auto slice on or off.

Arguments *autoSliceTF*

If the argument is true, auto slice is turned on. If false, it is turned off.

Returns Nothing.

dom.setButtonIncludeDownState

Description If the user is editing a button document, this function specifies whether to include the down state in a button.

Arguments *includeDownStateTF*

If the argument is true, the down state is included in the button. If false, it is not.

Returns Nothing.

dom.setButtonIncludeOverWhileDownState

Description If the user is editing a button document, this function specifies whether to include the over-while-down state in a button.

Arguments *includeOverWhileDownStateTF*

If the argument is true, the over-while-down state is included in the button. If false, it is not.

Returns Nothing.

dom.setButtonShowDownOnLoad

Description If the user is editing a button document, this function specifies whether to show the down-state-on-load in a button.

Arguments *showDownOnLoadTF*

If the argument is true, the down-state-on-load is shown in the button. If false, it is not.

Returns Nothing.

dom.setButtonOptions

Description Sets the button export options. If the user is editing a button, it sets options for the button being edited; if editing a normal document, it sets options for all selected buttons.

Arguments *exportOptions, URLString, altTagString, targetTagString, sliceName, statusMessage*

- The first argument is an ExportOptions object (see page 35).
- The second argument is a string that specifies the URL for the button(s).
- The third and fourth arguments specify the text for the button alt tag and target tag.
- The fifth argument (*sliceName*) is a string that specifies the name to be assigned to the slice that is associated with the button. If it is NULL, the slice is set to auto-name.
- The last argument is a string that specifies a status message to be displayed in the browser status line. If an empty string or NULL is passed, no status message is displayed.

Returns Nothing.

dom.setDefaultBrushAndFillColors()

Description Resets the document's brush and fill color to the default.

Arguments None.

Returns Nothing.

dom.setDefaultFillVector()

Description Sets the fill-vector on selection to the default.

Arguments None.

Returns Nothing.

dom.setDocumentCanvasColor()

Description Sets the canvas color of the document to the specified color.

Arguments *color*

The argument is a color string in the format "#rrggbb" or "#rrggbbaa".

Returns Nothing.

Example This command sets the canvas color to blue.

```
fw.getDocumentDOM().setDocumentCanvasColor("#0000ff");
```

dom.setDocumentCanvasSize()

Description Sets the document's canvas size to the specified rectangle.

Arguments *boundingRectangle*

- The first argument is a rectangle (see page 7) that specifies the new canvas size for the document, in pixels.
- Any items outside the specified rectangle are removed.

Returns Nothing.

Example Sets the canvas to a size of 200 by 200 pixels.

```
fw.getDocumentDOM().setDocumentCanvasSize({left:150, top:150, right:350,
bottom:350});
```

dom.setDocumentCanvasSizeToDocumentExtents()

Description Calculates the size of all items in the document, and resizes the document canvas to that size. This is the same behavior as Modify > Trim Canvas.

Arguments *growCanvasTF*

If the argument is true, the canvas can grow or shrink in size. If false, it can only shrink.

Returns Nothing.

Example This resizes the canvas to include all of the items in the document, enlarging the canvas if necessary.

```
fw.getDocumentDOM().setDocumentCanvasSizeToDocumentExtents(true);
```

Related Functions dom.setDocumentCanvasSizeToSelection()

dom.setDocumentCanvasSizeToSelection()

Description Calculates the size of all items in the selection, and resizes the document canvas to that size.

Arguments None.

Returns Nothing.

Related Functions dom.setDocumentCanvasSizeToDocumentExtents()

dom.setDocumentImageSize()

Description Scales the document to fit in the specified rectangle at the specified resolution.

Arguments *boundingRectangle, resolution*

- The first argument is a rectangle (see page 7) that specifies the size to which the document should be scaled.
- The second argument specifies the resolution (see page 7) for the scaled document.

Returns Nothing.

dom.setDocumentResolution()

Description Sets the resolution of the document.

Arguments *resolution*

The argument specifies the resolution (see page 7) for the document.

Returns Nothing.

dom.setExportOptions()

Description Sets the document export options.

Arguments *exportOptions*

The argument is an ExportOptions object (see page 35).

Returns Nothing.

dom.setExportSettings()

Description Sets the document export settings.

Arguments *exportSettings*

The argument is an ExportSettings object (see page 38).

Returns Nothing.

dom.setFill()

Description Sets the selection to the specified fill.

Arguments *fill*

The argument is a Fill object (see page 39).

Returns Nothing.

dom.setFillColor()

Description Changes the fill color of the selection to the specified color.

Arguments *color*

The argument is a color string in the format "#rrggbb" or "#rrggbbaa".

Returns Nothing.

dom.setFillEdgeMode()

Description Sets the edge type for selected items with fills.

Arguments *edgemode, featherAmt*

- Acceptable values for *edgemode* are "hard edge", "antialias", and "feather".
- The second argument (*featherAmt*) is an integer that specifies the number of pixels to feather. This value is ignored if *edgemode* is not "feather".

Returns Nothing.

dom.setFillNColorNTexture()

Description Sets the selection to the specified fill, fill color, and fill texture.

Arguments *fill, color, texture-name*

- The first argument is a Fill object (see page 39).
- The second argument is a color string in the format "#rrggbb" or "#rrggbbaa".
- The last argument is the name of the texture to be applied.

Returns Nothing.

Example This command sets the selected items to a linear fill with a feather edge and no texture.

```
fw.getDocumentDOM().setFillNColorNTexture({ category:"fc_Linear", ditherColors:[
"#000000", "#000000" ], edgeType:"antialiased", feather:10, gradient:{
name:"cn_WhiteBlack", nodes:[ { color:"#ffffff", position:0 }, { color:"#000000", position:1 }
] }, name:"fn_Normal", pattern:null, shape:"linear", stampingMode:"blend opaque",
textureBlend:0, webDitherTransparent:false }, "#666666", "Grain");
```

dom.setFillPlacement()

Description Sets the fill placement for selected items with fills.

Arguments *placement*

Acceptable values for *placement* are "top" and "bottom".

Returns Nothing.

dom.setFillVector()

Description Sets the fill-vectors of the selection to the specified absolute values.

Arguments *p1, p2, p3*

The arguments are points (see page 7) that specify the x, y coordinates of the three points to be used in calculating the fill vector.

Returns Nothing.

dom.setFillVectorStart()

Description Modifies the fill-vectors of the selection by moving the fill start to the specified point, and by then moving the two fill end handles to the same relative position.

Arguments *p1*

The argument is a point (see page 7) that specifies the x, y coordinates of the fill start and relative end handle placement to be used.

Returns Nothing.

dom.setGradientName()

Description Renames a gradient.

Arguments *currentName, newName*

- The first argument is a string that specifies the current name of the gradient.
- The second argument is a string that specifies what the new name of the gradient should be.

Returns Nothing.

dom.setGridOrigin()

Description Sets the grid origin for the document.

Arguments *gridOrigin*

The argument is a point (see page 7) that specifies the x, y coordinates that should be used for the document's grid origin.

Returns Nothing.

dom.setGridSize()

Description Sets the grid size for the document.

Arguments *gridSize*

The argument is a point (see page 7) that specifies the x, y coordinates that should be used for the document's grid size.

Returns Nothing.

dom.setGridColor()

Description Sets the color used to display the grid.

Arguments *gridColor*

The argument is a color string in the format "#rrggbb" or "#rrggbbaa".

Returns Nothing.

dom.setGuideColor()

Description Sets the color used to display normal (non-slice) guides. To set the color of slice guides, use dom.setSliceGuideColor().

Arguments *guideColor*

The argument is a color string in the format "#rrggbb" or "#rrggbbaa".

Returns Nothing.

dom.setGroupType()

Description Changes the group-type for groups in the selection.

Arguments *type*

Acceptable values for *type* are"normal", "mask to image", and "mask to path".

Returns Nothing.

dom.setHotspotAltTag()

Description For the hotspots and slices in the selection, sets the alt tag text to the specified value.

Arguments *whatToSet, altTagString*

- Acceptable values for *whatToSet* are "hotspots", "slices", and "hotspots and slices".
- The second argument is a string that specifies the text to be used for the alt tag.

Returns Nothing.

Example This command sets the alt tag of the selected slices to "This is my alt tag".

```
fw.getDocumentDOM().setHotspotAltTag("slices","This is my alt tag");
```

dom.setHotspotColor()

Description For the hotspots and slices in the selection, sets the color to the specified value.

Arguments *whatToSet, color*

- Acceptable values for *whatToSet* are "hotspots", "slices", and "hotspots and slices".
- The second argument is a color string in the format "#rrggbb" or "#rrggbbaa".

Returns Nothing.

Example This command sets the color of the selected hotspots to the specified value, which in this case is red.

```
fw.getDocumentDOM().setHotspotColor("hotspots", "#ff0000");
```

dom.setHotspotRectangle()

Description If the selection is a single hotspot or slice, this function moves or copies it to the specified location and size.

Arguments *boundingRectangle, makeCopyTF*

- The first argument is a rectangle (see page 7) that specifies the size of the new hotspot or slice.
- If the second argument is true, the selection is copied and resized instead of moved and resized.

Returns Nothing.

dom.setHotspotShape()

Description For the hotspots and slices in the selection, sets the shape to the specified value.

Arguments *whatToSet, shape*

- Acceptable values for *whatToSet* are "hotspots", "slices", and "hotspots and slices".
- Acceptable values for *shape* are "rectangle", "oval", and "polyline".

Returns Nothing.

dom.setHotspotTarget()

Description For the hotspots and slices in the selection, sets the target tag text to the specified value.

Arguments *whatToSet, targetTagString*

- Acceptable values for *whatToSet* are "hotspots", "slices", and "hotspots and slices".
- The second argument is a string that specifies the text to be used for the target tag.

Returns Nothing.

Example This command sets the currently selected slices to link to the parent window.

```
fw.getDocumentDOM().setHotspotTarget("slices", "_parent");
```

dom.setHotspotText()

Description For the hotspots and slices in the selection, sets the hotspot text to the specified value.

Arguments *whatToSet, textString, urlToMatch, updateAttributesTF*

- Acceptable values for *whatToSet* are "hotspots", "slices", and "hotspots and slices".
- The second argument is a string that specifies the text to be used for the hotspot or slice.
- The third argument (*urlToMatch*) is a string that specifies a URL that is already assigned to one or more hotspots in the document. If this value is not NULL, then all hotspots or slices in the document that have *urlToMatch* as their URL have their URL changed to *textString*. Note: the URLs of both selected and unselected hotspots or slices are changed.
- If the last argument (*updateAttributesTF*) is true, hotspots that are being changed inherit the color, target, and alt tag text that were most recently associated with the new text value. For example, suppose *textString* is "http://www.mywebsite.com", and the last time "http://www.mywebsite.com" was used, it was used with a color of blue, a target of none, and an alt tag of "Link to My Home Page". If *updateAttributesTF* is true, then any hotspot or slice whose text is now being changed to "http://www.mywebsite.com" will also have a color of blue, a target of none, and alt tag text of "Link to My Home Page".

Returns Nothing.

Example This command creates a slice and inserts the HTML text "I am HTML text".

```
fw.getDocumentDOM().setHotspotText("Slice ","I am HTML text", null, true);
```

dom.setLayerLocked()

Description Locks or unlocks one or all layers on the specified frame.

Arguments *layerIndex, frameIndex, lockTF, allLayersTF*

- The first argument is a zero-based integer that specifies the layer to be locked or unlocked. To specify the current layer, pass –1. (To lock/unlock all layers on a frame, use the *allLayersTF* argument.)
- The second argument is a zero-based integer that specifies the frame containing the layer to be locked or unlocked. To specify the current frame, pass –1.
- If the third argument (*lockTF*) is true, the layer is locked. If false, it is unlocked.
- If the last argument (*allLayersTF*) is true, all layers on the specified frame are locked or unlocked, and any value passed for *layerIndex* is ignored.

Returns Nothing.

Example This command locks all of the layers on the first frame.

```
fw.getDocumentDOM().setLayerLocked(1, 0, true, true);
```

dom.setLayerName()

Description Renames the specified layer. Layers aren't required to have unique names, so no duplicate checking is performed.

Arguments *layerIndex, layerName*

- The first argument is a zero-based integer that specifies the layer to be renamed. To specify the current layer, pass –1.
- The second argument is a string that specifies the new name for the layer.

Returns Nothing.

dom.setLayerSharing()

Description Changes the "shared layer" status of a layer.

Arguments *layerIndex, sharedStatus, unshareCopiesToAllFramesTF, warnUserTF*

- The first argument is a zero-based integer that specifies the layer to be shared or not shared. To specify the current layer, pass –1.
- Acceptable values for *sharedStatus* are "shared" and "not shared".
- The third argument (*unshareCopiesToAllFramesTF*) is used only if *sharedStatus* is "not shared" and the document has multiple frames. If these conditions are met and *unshareCopiesToAllFramesTF* is true, then the items on the layer are duplicated to all frames of the layer; if false, the items are placed only on the current frame.
- If the last argument is true and *unshareCopiesToAllFramesTF* is enabled, the user is asked to confirm that data on other frames can be overwritten. If false, data on other frames of the layer is overwritten without warning.

Returns Nothing.

Example This command sets the selected layer to "Shared" and displays a warning that data loss is possible.

```
fw.getDocumentDOM().setLayerSharing(-1, "shared", false, true);
```

dom.setLayerVisible()

Description Shows or hides a layer on the specified frame.

Arguments *layerIndex, frameIndex, showTF, allLayersTF*

- The first argument is a zero-based integer that specifies the layer that should be shown or hidden. To specify the current layer, pass –1. (To show/hide all layers on a frame, use the *allLayersTF* argument.)
- The second argument is a zero-based integer that specifies the frame containing the layer to be shown or hidden. To specify the current frame, pass –1.
- If the third argument (*showTF*) is true, the layer is visible. If false, it is hidden.
- If the last argument (*allLayersTF*) is true, all layers on the specified frame are shown or hidden, and any value passed for *layerIndex* is ignored.

Returns Nothing.

dom.setMatteColor()

Description Sets or removes the document's matte color used for exporting.

Arguments *useMatteColorTF, matteColor*

- If the first argument is true, the document's matte color is set to the value specified by *matteColor*. If false, any matte color is removed from the document, and the second argument is ignored.
- The second argument is a color string in the format "#rrggbb" or "#rrggbbaa".

Returns Nothing.

Example This command sets the matte color to the specified value, which in this case is blue.

```
fw.getDocumentDOM().setMatteColor(true, "#0033ff");
```

dom.setPixelMask()

Description If Fireworks is in image edit mode, this function sets the pixel-selection mask of the current image to the specified mask.

Arguments *mask, howToCombineMasks*

- The first argument is a mask variable (see page 6) that specifies the mask to be applied. If *mask* is NULL, then any existing pixel selection mask is removed.
- If there was previously a mask and the new mask is also not NULL, then *howToCombineMasks* specifies how the two masks should be combined. Acceptable values for *howToCombineMasks* are "replace", "add", "subtract", and "intersect".

Returns Nothing.

dom.setOnionSkinning()

Description Sets the onion skin display options for the document.

Arguments *before, after*

- The arguments are integers that specify the number of frames to display before and after the current one.
- To disable onion skinning, pass zero for both arguments.
- To enable onion skinning for all frames, pass zero for the first argument and a very large number for the second argument (e.g., 99999).

Returns Nothing.

Example This command turns on onion skinning two frames before the selected frame and zero frames after it.

```
fw.getDocumentDOM().setOnionSkinning(2, 0);
```

dom.setOpacity()

Description Sets the opacity of the selection to the specified value.

Arguments *opacity*

The argument is a float variable between 0 and 100, inclusive.

Returns Nothing.

Example This command sets the select item to an opacity of 55%.

```
fw.getDocumentDOM().setOpacity(55);
```

dom.setQuadrangle()

Description Transforms the selection within a specified bounding quadrangle. The effect is the same as performing a Transform operation within Fireworks, and then replaying the Transform step from the History panel while other items are selected.

Arguments *pTopLeft, pTopRight, pBottomRight, pBottomLeft, options*

- The first four arguments are points (see page 7) that specify the x,y coordinates of the top left, top right, bottom right, and bottom left points of the bounding rectangle.
- Acceptable values for *options* are "transformAttributes", "autoTrimImages", and "autoTrimImages transformAttributes".

Returns Nothing.

Example This command transforms the selection as specified.

```
fw.getDocumentDOM().setQuadrangle({x:-0.300884962, y:0.207964599}, {x:1,
y:0.207964599}, {x:1, y:0.792035401}, {x:-0.300884962, y:0.792035401},
"autoTrimImages transformAttributes");
```

dom.setSelectionBounds()

Description Moves and resizes the selection in a single operation.

Arguments *boundingRectangle, opts*

- The first argument is a rectangle (see page 7) that specifies the new location and size of the selection.
- Acceptable values for *opts* "transformAttributes", "autoTrimImages", and "autoTrimImages transformAttributes".

Returns Nothing.

dom.setSliceAutonaming()

Description If a single slice is selected, this function turns slice autonaming on or off for the slice.

Arguments *autonameTF*

If the argument is true, autonaming is turned on for the slice. If false, it is turned off.

Returns Nothing.

dom.setSliceExportOptions()

Description Sets the export options for the selected slices.

Arguments *exportOptions*

The argument is an ExportOptions object (see page 35).

Returns Nothing.

dom.setSliceFilename()

Description If a single slice is selected, this function turns off autonaming for the slice and sets its filename to the specified URL.

Arguments *fileURL*

The argument is a fileURL specifying the name to be given to the slice.

Returns Nothing.

dom.setSliceGuideColor()

Description Sets the color used to display slice guides. To set the color of normal guides, use dom.setGuideColor().

Arguments *color*

The argument is a color string in the format "#rrggbb" or "#rrggbbaa".

Returns Nothing.

dom.setShowSliceOverlay()

Description Specifies whether the slice overlay is visible or not.

Arguments *showTF*

If the argument is true, the slice overlay is visible. If false, it is not.

Returns Nothing.

dom.setSliceHtml()

Description If a single slice is selected, this function sets the slice's HTML text.

Arguments *htmlText*

The argument is a string specifying the HTML text for the slice.

Returns Nothing.

dom.setSliceIsHtml()

Description Sets the selected slices as HTML or Image.

Arguments *htmlTF*

If the argument is true, sets the slices as HTML. If false, sets them to Image.

Returns Nothing.

dom.setShowEdges()

Description Specifies whether "show edges" is on or off.

Arguments *showEdgesTF*

If the argument is true, "show edges" is turned on. If false, it is turned off.

Returns Nothing.

dom.setShowGammaPreview()

Description Specifies whether "Preview Gamma" is on or off.

Arguments *previewGammaTF*

If the argument is true, "Preview Gamma" is turned on. If false, it is turned off.

Returns Nothing.

dom.setShowGrid()

Description Specifies whether the grid is visible.

Arguments *showTF*

If the argument is true, the grid is visible. If false, it is not.

Returns Nothing.

dom.setShowGuides()

Description Specifies whether normal guides are visible.

Arguments *showTF*

If the argument is true, the normal guides are visible. If false, they are not.

Returns Nothing.

dom.setShowRulers()

Description Specifies whether rulers are visible.

Arguments *showTF*

If the argument is true, the rulers are visible. If false, they are not.

Returns Nothing.

dom.setShowSliceGuides()

Description Specifies whether slice guides are visible.

Arguments *showTF*

If the argument is true, the slice guides are visible. If false, they are not.

Returns Nothing.

dom.setSnapToGrid()

Description Specifies whether tools snap to grids.

Arguments *snapTF*

If the argument is true, the tools snap to grids. If false, they do not.

Returns Nothing.

dom.setSnapToGuides()

Description Specifies whether tools snap to guides.

Arguments *snapTF*

If the argument is true, the tools snap to all guides. If false, they do not.

Returns Nothing.

dom.setSymbolProperties()

Description Sets the name and symbol type of the specified symbol.

Arguments *currentName, symbolType, newName*

- The first argument specifies the current name of the symbol in the Library. If more than one master exists with a name of *currentName*, then only the first master is changed. If NULL is passed in for *currentName*, then the name property is set for all selected symbols in the Library (not document).
- Acceptable values for *symbolType* are "graphic" and "button".
- The last argument specifies the new name for the symbol.

Returns Nothing.

dom.setTextAlignment()

Description Sets the alignment of the selected text items to the specified setting.

Arguments *alignment*

Acceptable values for *alignment* are "left", "center", "right", "justify", "stretch", "vertical left", "vertical center", "vertical right", "vertical justify", and "vertical stretch".

Returns Nothing.

dom.setTextAutoKern()

Description Specifies whether auto-kerning is on or off for the selected text items.

Arguments *kernTF*

If the argument is true, auto-kerning is on for the selected text items. If false, it is not.

Returns Nothing.

dom.setTextFlow()

Description Sets the horizontal flow direction of the selected text items.

Arguments *flowDirection*

Acceptable arguments for *flowDirection* are "left to right" and "right to left".

Returns Nothing.

dom.setTextOnPathMode()

Description Sets the mode of the selected text-on-a-path items to the specified value.

Arguments *mode*

Acceptable values for *mode* are "rotate", "vertical", "skew vertical", and "skew horizontal".

Returns Nothing.

dom.setTextOnPathOffset()

Description Sets the offset for the selected text-on-a-path to the specified distance.

Arguments *offset*

The argument is a float value that specifies the offset distance, in pixels.

Returns Nothing.

dom.setTextOrientation()

Description Sets the horizontal/vertical text orientation of the selected text items.

Arguments *orientation*

Acceptable values for *orientation* are "horizontal left to right", "vertical right to left", "horizontal right to left", and "vertical left to right".

Returns Nothing.

dom.setTextRuns()

Description Replaces the text in the selected text blocks with the styled text described by the object passed.

Arguments *textRuns*

The argument is a TextRuns object (see page 48).

Returns Nothing.

dom.setTransformMode()

Description Sets the transform mode for the selected text or instance items, or both.

Arguments *mode*

Acceptable values for *mode* are "paths" and "pixels".

Returns Nothing.

dom.setTextAntiAliasing()

Description Sets the antialiasing level for the selected blocks of text.

Arguments *level*

Acceptable values for *level* are "crisp", "smooth", and "strong".

Returns Nothing.

dom.setTextRectangle()

Description Changes the bounding rectangle for the selected text item to the specified size. This function causes the text to re-flow inside the new rectangle; the text item is not scaled or transformed. Text that does not fit into the new rectangle is not displayed.

Arguments *boundingRectangle*

The argument is a rectangle (see page 7) that specifies the new size within which the text item should flow.

Returns Nothing.

dom.setTextRectangleAuto()

Description Recalculates the bounding rectangle for the selected text item, setting the rectangle to the smallest box that encloses the text.

Arguments None.

Returns Nothing.

Related Functions dom.setTextRectangleAutoFromPoint()

dom.setTextRectangleAutoFromPoint()

Description Performs the same function as dom.setTextRectangleAuto(), but lets you pass a point to specify where the rectangle should be located.

Arguments *anchorPoint*

The argument is a point (see page 7) that specifies the x, y coordinates of the location at which the text box should be anchored. How the point is used depends on the left-to-right/up-to-down orientation of the text flow in the text block.

- Left-justified horizontal text is placed with its top and left edges at *anchorPoint*, and the text expands to the right.
- Centered horizontal text is centered horizontally around *anchorPoint*, and expands equally to the left and right.
- Centered vertical text is centered vertically around *anchorPoint*, and expands equally up and down.

Returns Nothing.

Related Functions dom.setTextRectangleAuto()

dom.showAllHidden()

Description Shows all the items that were hidden by using dom.hideSelection().

Arguments None.

Returns Nothing.

dom.splitPaths()

Description Split the selected paths. Compound paths are split into separate contours.

Arguments None.

Returns Nothing.

dom.swapBrushAndFillColors()

Description Swaps the current brush color and current fill color. This function has no effect on any selected items.

Arguments None.

Returns Nothing.

dom.transformSelection()

Description Transforms the selection using the specified three-by-three matrix.

Arguments *matrix, options*

- The first argument is a three-by-three transformation matrix (see page 7).
- Acceptable values for *options* are "transformAttributes", "autoTrimImages", and "autoTrimImages transformAttributes".

Returns Nothing.

dom.tween()

Description Tweens between the two selected instances.

Arguments *numSteps, distributeTF*

- The first argument is an integer that specifies how many new instances are generated.
- If *distributeTF* is true, the new instances are distributed to frames.

Returns Nothing.

dom.updateSymbol()

Description Updates the specified linked symbol.

Arguments *name*

The argument specifies the name of the symbol in the Library. If more than one symbol exists with a name of *name*, then only the first symbol with that name is updated. If NULL is passed in for *name*, then all selected linked symbols in the Library (not document) are updated.

Returns Nothing.

dom.undo()

Description Undoes the undoable step that was most recently performed. Most (but not all) JavaScript functions cause an undoable action to be executed.

Arguments None.

Returns Nothing.

dom.ungroup()

Description Ungroups any grouped items in the selection. To group items, use dom.group().

Arguments None.

Returns Nothing.

History panel functions

fw.historyPalette.clearSteps()

Description Clears the undo/redo stack.

Arguments None.

Returns Nothing.

fw.historyPalette.copySteps()

Description Copies the selected history steps into the Clipboard.

Arguments *array of indices*

The argument is a zero-based array that specifies which steps from the History panel should be copied. If it is NULL, the currently selected steps are used.

Returns Nothing.

fw.historyPalette.getSelection()

Description Determines which steps in the History panel are selected.

Arguments None.

Returns A zero-based array representing which History panel steps are selected.

fw.historyPalette.getStepCount()

Description Returns the number of steps in the History panel.

Arguments None.

Returns The number of steps in History panel (not a zero-based value).

fw.historyPalette.getStepsAsJavaScript()

Description Gets the JavaScript equivalent of the selected steps.

Arguments *array of indices*

The argument is a zero-based array that specifies which steps from the History panel should be returned as JavaScript. If the argument is NULL, the currently selected steps are returned.

Returns JavaScript string

Related Functions fw.historyPalette.replaySteps()

fw.historyPalette.getUndoState()

Description Returns a string that indicates the current undo state, to be used for later calls to fw.historyPalette.setUndoState().

Arguments None.

Returns String to be used with fw.historyPalette.setUndoState(). This string is designed to be used internally by Fireworks only, and may change format in the future. Therefore, do not try to parse this string or to construct a custom string to pass to fw.historyPalette.setUndoState().

fw.historyPalette.replaySteps()

Description Gets the JavaScript equivalent of the selected steps and executes them.

Arguments *array of indices*

The argument is a zero-based array that specifies which steps from the History panel should be returned as JavaScript and then executed. If the argument is NULL, the currently selected steps are used.

Returns JavaScript string

Related Functions fw.historyPalette.getStepsAsJavaScript()

fw.historyPalette.saveAsCommand()

Description Gets the JavaScript equivalent of the selected steps and saves them as a JSF command file.

Arguments *array of indices, {filename}*

- The first argument indicates which steps from the History panel should be saved. For example, to save the first, third, and sixth step in the History panel, pass [0, 2, 5]. If this argument is NULL, the currently selected steps are used.
- The second argument is an optional string that specifies a name for the JSF command file. It can be any string, including a fileURL. If *filename* is omitted or NULL, the user is prompted for a filename. If *filename* is not a fileURL, then the file is saved in the Fireworks 3/Settings/Commands folder with the specified filename.

Returns Nothing.

fw.historyPalette.setSelection()

Description Sets the portion of the History panel which is selected.

Arguments *array of indices*

The argument specifies which steps in the History panel are selected. Values are zero-based. For example, to select the first, third, and sixth step in the History panel, pass [0, 2, 5].

Returns Nothing.

fw.historyPalette.setUndoState()

Description Performs the correct number of undo or redo operations to arrive at selected state.

Arguments *undoStateString*

The argument is the string returned by fw.historyPalette.getUndoState().

Returns Nothing.

Fireworks functions

In Fireworks 3, fw is synonymous with fireworks. Thus all of the methods of the fireworks object can be referred to as fireworks.*functionName()* or as fw.*functionName()*.

fw.browseDocument()

Description Opens the user's primary browser and displays the specified URL.

Arguments *URL*

The argument is the URL of the page to be displayed in the browser. Any legal URL (including http://, ftp://, and so on) can be passed. Fireworks does not check this argument for syntax; if you pass an illegal value, the browser will fail to open the URL.

Returns Nothing.

fw.browseForFileURL()

Description Displays an Open or Save dialog box for the user.

Arguments *browseType, title, previewArea*

- Acceptable values for *browseType* are "open", "select", and "save". The first two display an Open dialog box; they are both acceptable here for compatibility with Dreamweaver. The third value displays a Save dialog box.
- The second and third arguments are ignored by Fireworks, but are accepted for compatibility with Dreamweaver.

Returns The fileURL selected by the user, or NULL if the dialog box was canceled.

fw.browseForFolderURL()

Description Displays a dialog box that lets a user select a particular directory.

Arguments {*title*}, {*startFolder*}

- The first argument is an optional string that specifies a title for the dialog box that is displayed. If it is omitted or NULL, a default title is displayed.
- The second argument is an optional string that serves as the root directory for the dialog box that is displayed. If it is omitted or NULL, the browse dialog box displays an unspecified directory, depending on your system configuration. Generally, it is the last directory used.

fw.checkFwJsVersion()

Description Checks the JavaScript API for incompatibilities.

Arguments *version*

The argument is a long variable that is reserved for future use; only a value of 0 is supported at this time. To use this function, put a call to *fw.checkFwJsVersion(0)* in your script.

Returns Nothing.

fw.closeDocument()

Description Closes the specified document.

Arguments *document, {promptToSaveChangesTF}*

- The first argument is a Document object (for example, fw.documents[2]) that specifies the document to close.
- If *promptToSaveChangesTF* is true or omitted, and the document has been changed since the last time it was saved, the user is prompted to save any changes to the document. If *promptToSaveChangesTF* is false, the user is not prompted, and any changes to the document are discarded.

Returns Nothing.

fw.createDocument()

Description Opens a new document and selects it. Values for size, resolution, and color are the same as the current defaults. To specify values other than the defaults, use fw.createFireworksDocument().

Arguments None.

Returns The Document object (see page 15) for the newly created document.

fw.createFireworksDocument()

Description Opens a new document and selects it. Values for size, resolution, and color are explicitly specified. To open a new document with default values, use fw.createDocument().

Arguments *size, res, backgroundColor*

- The first argument is a point whose x value specifies the document's width and whose y value specifies the document's height. Both values are pixels.
- The second argument specifies the resolution (see page 7) for the scaled document.
- The last argument is a color string in the format "#rrggbb" or "#rrggbbaa".

Returns The Document object (see page 15) for the newly created document.

Example This command creates a new document 500x500 pixels in size, with a resolution of 72 dpi and a solid white background color.

```
fw.createFireworksDocument({x:500,y:500},{pixelsPerUnit:72,units:"inch"}, "#ffffff");
```

fw.exportDocumentAs()

Description Exports the specified document to the specified file.

Arguments *document, fileURL, exportOptions*

- The first argument is a Document object (for example, fw.documents[2]) that specifies the document to be exported. If *document* is NULL, the active document is exported.
- The second argument is a fileURL that specifies the filename for the exported file. If *fileURL* is NULL, the Save As dialog box is displayed.
- The last argument is an ExportOptions object (see page 35). If *exportOptions* is NULL, the document's current export options are used. If the file format specified by *exportOptions* conflicts with the file format specified by *fileURL*, then the extension of *fileURL* is changed to match the format specified by *exportOptions*.

Returns Nothing.

fw.findNext()

Description Finds the next instance of the current search string and selects that section of the document. To begin a search, use fw.setUpFindReplace().

Arguments None.

Returns The number of items replaced if the search is completed, or –1 if there are items in the document remaining to be searched.

fw.getDocumentDOM()

Description Returns the Document object (see page 15) for the active document.

Arguments {*which-string*}

The argument is an optional string included for compatibility with Dreamweaver. If specified here, it must be "document".

Returns The Document object for the active document, or NULL if no document is open.

fw.getDocumentPath()

Description Gets the path and filename of the specified document.

Arguments *document*

The argument is a Document object (for example, fw.documents[2]) that specifies the document whose path and filename should be retrieved. If *document* is NULL, information about the active document is retrieved.

Returns The fileURL for the document if it has been saved, or an empty string if it has not yet been saved.

fw.getFloaterGroupings()

Description Gets an array of arrays that indicates the tab-grouping of all the panels (even hidden ones).

Arguments None.

Returns An array looking something like this:

```
[ [ "stroke", "fill", "effect" ], [ "layers", "frames", "object" ], [ "mixer", "options", "swatches", "info" ], [ "styles", "library" ], [ "find", "project log" ], [ "url" ], [ "optimize", "optimized colors" ], [ "behaviors" ], [ "history" ] ]
```

fw.getFloaterPosition()

Description Gets the screen position and size of the specified panel.

Arguments *panelName*

Acceptable values for *panelName* are "find", "project log", "object", "info", "url", "effect", "history", "mixer", "fill", "stroke", "swatches", "layers", "frames", "behaviors", "optimize", "library", "styles", "optimized colors", "options", and "toolbox".

Returns A rectangle (see page 7) that specifies the bounds of the panel.

fw.getFloaterVisibility()

Description Checks to see if a specified panel is visible.

Arguments *panelName*

Acceptable values for *panelName* are "find", "project log", "object", "info", "url", "effect", "history", "mixer", "fill", "stroke", "swatches", "layers", "frames", "behaviors", "optimize", "library", "styles", "optimized colors", "options", and "toolbox".

Returns true if the specified panel is visible, false otherwise.

fw.getHideAllFloaters()

Description Returns the hidden or visible status of the panels.

Arguments None.

Returns true if the panels are hidden, false otherwise.

fw.openDocument()

Description Opens the specified document.

Arguments {*fileURL*}

The argument is a fileURL for the file to be opened. If *fileURL* is omitted or NULL, the Open Document dialog box is displayed.

Returns Nothing.

fw.quitApplication()

Description Quits Fireworks, prompting the user to save any changed documents.

Arguments None.

Returns Nothing.

fw.replace()

Description Verifies that the selection matches the current search string and replaces it with the replacement string.

Arguments None.

Returns The number of items replaced, or –1 if there are items in the document remaining to be searched.

Related Functions fw.setUpFindReplace()

fw.replaceAll()

Description Performs a Replace All operation on the active document, using the current search and replacement strings.

Arguments None.

Returns The number of items replaced, or –1 if the find is not yet complete.

Related Functions fw.setUpFindReplace()

fw.revertDocument()

Description Reverts the specified document to its previously saved version.

Arguments {*document*}

The argument is a Document object (for example, fw.documents[2]) that specifies the document to be reverted. If *document* is omitted or NULL, the active document is reverted.

Returns Nothing.

fw.runScript()

Description Execute a JavaScript file.

Arguments *filename*

The argument is the name of the script file to be executed. If *filename* is not a fileURL (that is, it does not begin with "file:///"), it is assumed to be the name of a file in the Fireworks 3/Settings/Commands folder.

Returns Result of script.

Example This command runs a script found in the Commands folder called "Align Center to Document".

```
fw.runScript("Align Center to Document.jsf");
```

fw.saveAll()

Description Saves all open documents, displaying the Save As dialog box for any documents that have not previously been saved.

Arguments None.

Returns Nothing.

fw.saveDocument()

Description Saves the specified document as a native Fireworks PNG file with the specified name. To save a document to another format, such as GIF or JPEG, use fw.exportDocumentAs().

Arguments *document, {fileURL}*

- The first argument is a Document object (for example, fw.documents[2]) that specifies the document to be saved. If *document* is NULL, the active document is saved.
- The second argument is the fileURL of the saved document. If *fileURL* is NULL or omitted, the document is saved with its current name; if the document has not yet been saved, the Save As dialog box is displayed.

Returns Nothing.

fw.saveDocumentAs()

Description Displays the Save As dialog box for the specified document, allowing it to be saved as a native Fireworks PNG file with the specified name. To save a document to another format, such as GIF or JPEG, use fw.exportDocumentAs().

Arguments *document*

The argument is a Document object (for example, fw.documents[2]) that specifies the document to be saved. If *document* is NULL, the active document is saved.

Returns The fileURL for the saved document, or NULL if the dialog box was canceled.

fw.saveDocumentCopyAs()

Description Saves a copy of the specified document as a native Fireworks PNG file with the specified name. To save a document to another format, such as GIF or JPEG, use fw.exportDocumentAs().

Arguments *document, fileURL*

- The first argument is a Document object (for example, fw.documents[2]) that specifies the document to be saved. If *document* is NULL, the active document is saved.
- The second argument is a fileURL that specifies the filename for the saved file. If *fileURL* is NULL, the Save As dialog box is displayed.

Returns The fileURL for the saved document, or NULL if the dialog box was canceled.

fw.saveJsCommand()

Description Saves the specified string of JavaScript code as a JSF command file.

Arguments *jscode, filename*

- The first argument specifies the string of code to be saved as a JSF command file.
- The second argument specifies the name in which the file should be saved. If *filename* is not a fileURL (that is, it does not begin with "file:///"), the file is saved in the Fireworks 3/Settings/Commands folder.

Returns Nothing.

fw.setActiveWindow()

Description Sets the specified document to be the active document.

Arguments *document, {trueFalse}*

- The first argument is a Document object (for example, fw.documents[2]) that specifies which document should be made active.
- The second argument is optional, and is ignored by Fireworks. It is included only for Dreamweaver compatibility.

Returns Nothing.

Example This command makes the fourth document the active document.

```
fw.setActiveWindow(fw.documents[3]);
```

fw.setFloaterGrouping()

Description Moves the specified panel into another specified panel, changing it to a tab within that panel. This is the same behavior as dragging a tab from one panel to another, or to its own panel.

Arguments *panelNameToMove, panelNameToReceive*

- The first argument is a lowercase string that specifies the panel to be moved.
- The second argument is a lowercase string that specifies the panel into which *panelNameToMove* should be moved. If *panelNameToReceive* is NULL, then the *panelNameToMove* is moved into a new panel by itself.

Returns Nothing.

Example This command moves the Stroke tab from its current location into the panel named Object. Even though the panel name may be capitalized onscreen, it must be passed as lower case.

```
fw.setFloaterGrouping("stroke", "object");
```

fw.setFloaterPosition()

Description Sets the position and size of a panel.

Arguments *panelName, boundingRectangle*

- Acceptable values for *panelName* are "find", "project log", "object", "info", "url", "effect", "history", "mixer", "fill", "stroke", "swatches", "layers", "frames", "behaviors", "optimize", "library", "styles", "optimized colors", "options", and "toolbox".
- The second argument is a rectangle (see page 7) that specifies the size of the panel. Some panels ignore the specified size, but place the top left corner of the panel at the top left location of the specified rectangle.

Returns Nothing.

fw.setFloaterVisibility()

Description Shows or hides the specified panel.

Arguments *panelName, visibleTF*

- Acceptable values for *panelName* are "find", "project log", "object", "info", "url", "effect", "history", "mixer", "fill", "stroke", "swatches", "layers", "frames", "behaviors", "optimize", "library", "styles", "optimized colors", "options", and "toolbox".
- If the second argument is true, the specified panel is visible. If false, it is hidden.

Returns Nothing.

fw.setHideAllFloaters()

Description Shows or hides the panels. This behavior is the same as the tab key functionality.

Arguments *hideTF*

If the argument is true, panels are hidden. If false, they are visible.

Returns Nothing.

fw.setUpFindReplace()

Description Sets up a search.

Arguments *findSpec*

The argument is a Find object (see page 20).

Returns Nothing.

fw.toggleFloater()

Description Shows, hides, or makes topmost the specified panel.

- If the panel is not visible, makes it visible and topmost.
- If the panel is topmost, hides it.
- If the panel is visible but not topmost, makes it topmost.

Arguments *panelName*

Acceptable values for *panelName* are "find", "project log", "object", "info", "url", "effect", "history", "mixer", "fill", "stroke", "swatches", "layers", "frames", "behaviors", "optimize", "library", "styles", "optimized colors", "options", and "toolbox".

Returns Nothing.

Using the addBehavior() function

The syntax for dom.addBehavior() (see page 59) is:

fw.getDocumentDOM().addBehavior(*action, event, eventindex*);

where the first argument is a string that specifies the behavior to be added. The information in this section describes the acceptable values for the first argument passed to dom.addBehavior().

MM_simpleRollover

Description Adds a simple rollover behavior.

Arguments None.

Example fw.getDocumentDOM().addBehavior("MM_simpleRollover()", "onMouseOver", -1);

MM_swapImage

Description Adds a swap image behavior.

Arguments *target, swapFrame, fileName, preload, restoreOnMouseOut*

- The first argument specifies the slice that the behavior is attached to. Pass –1 for this value; all other values are used internally by Fireworks.
- The second argument (*swapFrame*) is a zero-based integer that specifies the frame to swap. To use *fileName* as a URL, pass –1 here.
- The third argument (*fileName*) specifies the frame or file to be swapped. If you specified a frame to use in *swapFrame*, pass an empty text string here. If you want to specify a file name and you passed –1 for *swapFrame*, pass the string for the relative URL of the image here.
- The fourth argument (*preload*) is a binary value that specifies whether to preload the swapped image (pass 1) or not (pass 0).
- The last argument (*restore*) is a binary value that specifies whether to restore on mouse out (pass 1) or not (pass 0).

Example fw.getDocumentDOM().addBehavior("MM_swapImage(-1,1,\"\",1,1)", "onMouseOver", -1);

MM_swapImgRestore

Description Adds a swap image restore behavior.

Arguments None.

Example fw.getDocumentDOM().addBehavior("MM_swapImgRestore()", "onMouseOut", -1);

MM_statusMessage

Description Sets a status bar message.

Arguments *message*

The argument is a string that specifies the status message to be displayed.

Example

```
fw.getDocumentDOM().addBehavior("MM_statusMessage(\"Status Message!\")",
"onMouseOver", -1);
```

MM_nbGroup [image]

Description Sets a Navigation Bar image behavior.

Arguments *type, downHighlight, initiallyDown*

- Pass "all" for *type*.
- The second argument (*downHighlight*) is a binary value that specifies whether the image should be highlighted on mouse down (pass 1) or not (pass 0).
- The third argument (*initiallyDown*) is a binary value that specifies whether the image should initially appear as down (pass 1) or not (pass 0).

Example

```
fw.getDocumentDOM().addBehavior("MM_nbGroup(\'all\',1,0)", "onMouseOver", -1);
```

MM_nbGroup [highlight]

Description Sets a Navigation Bar highlight behavior.

Arguments *type, target, swapFrame, fileName, preload, downHighlight, downHighlightFrame, downHighlightFilename*

- Pass "over" for *type*.
- The second argument (*target*) specifies the slice that the behavior is attached to. Pass –1 for this value; all other values are used internally by Fireworks.
- The third argument (*swapFrame*) is a zero-based integer that specifies the frame to swap. To use *fileName* as a URL, pass –1 here.
- The fourth argument (*fileName*) specifies the frame or file to be swapped. If you specified a frame to use in *swapFrame*, pass an empty text string here. If you want to specify a file name and you passed –1 for *swapFrame*, pass the string for the relative URL of the image here.
- The fifth argument (*preload*) is a binary value that specifies whether to preload the swapped image (pass 1) or not (pass 0).
- The sixth argument (*downHighlight*) is a binary value that specifies whether an image should be used for highlighting on mouse down (pass 1) or not (pass 0). If you pass 1, use the next two arguments to specify the frame or image to be used.
- The seventh argument (*downHighlightFrame*) is a zero-based integer that specifies the frame to use as a highlight image. To use *downHighlightFilename* as a URL, pass –1 here.
- The last argument (*downHighlightFilename*) specifies the frame or file to be used as the highlight image. If you specified a frame to use in *downHighlightFrame*, pass an empty text string here. If you want to specify a file name and you passed –1 for *downHighlightFrame*, pass the string for the relative URL of the image here.

Example

```
fw.getDocumentDOM().addBehavior("MM_nbGroup(\'over\',-1,1,\"\",1,0,3,\"\")",
"onMouseOver", -1);
```

MM_nbGroup [down]

Description Sets a Navigation Bar Down behavior.

Arguments *type, barName, target, swapFrame, fileName, preload*

- Pass "down" for *type*.
- Pass "navbar1" for the name of the Navigation Bar.
- The third argument (*target*) specifies the slice that the behavior is attached to. Pass –1 for this value; all other values are used internally by Fireworks.
- The fourth argument (*swapFrame*) is a zero-based integer that specifies the frame to swap. To use *fileName* as a URL, pass –1 here.
- The fifth argument (*fileName*) specifies the frame or file to be swapped. If you specified a frame to use in *swapFrame*, pass an empty text string here. If you want to specify a file name and you passed –1 for *swapFrame*, pass the string for the relative URL of the image here.
- The last argument (*preload*) is a binary value that specifies whether to preload the swapped image (pass 1) or not (pass 0).

Example

```
fw.getDocumentDOM().addBehavior("MM_nbGroup(\'down\',\'navbar1\',-1,2,\"\",1)",
"onClick", -1);
```

MM_nbGroup [out]

Description Sets a Navigation Bar Restore behavior.

Arguments *type*

Pass "out" for *type*.

```
fw.getDocumentDOM().addBehavior("MM_nbGroup(\'out\')", "onMouseOut", -1);
```