Photoshop® CS JavaScript Reference Guide

Adobe Developer Support
345 Park Avenue
San Jose, CA 95110-2704
408-536-9000
FaxYI: 206-628-5737
ada@adobe.com

http://partners.adobe.com

October 2003

Adobe® Photoshop® CS JavaScript Reference Guide

Table of Contents

1

Introduction

This reference guide describes the objects and commands in the Adobe® Photoshop® CS JavaScript type library. In addition to this Introduction, it includes the following sections:

- Chapter 2 -- Describes how to create user interface elements using JavaScript
- Chapter 3 -- Describes how to use File and Folder objects to abstract platform interfaces
- Chapter 4 -- Describes Debugging tools and techniques
- Chapter 5 -- Describes how to use Action Manager within JavaScripts
- Chapter 6 -- Describes the classes, properties and methods of the JavaScript interface
- Chapter 7 -- Describes the fundamental syntax of JavaScript

JavaScript Sample Code

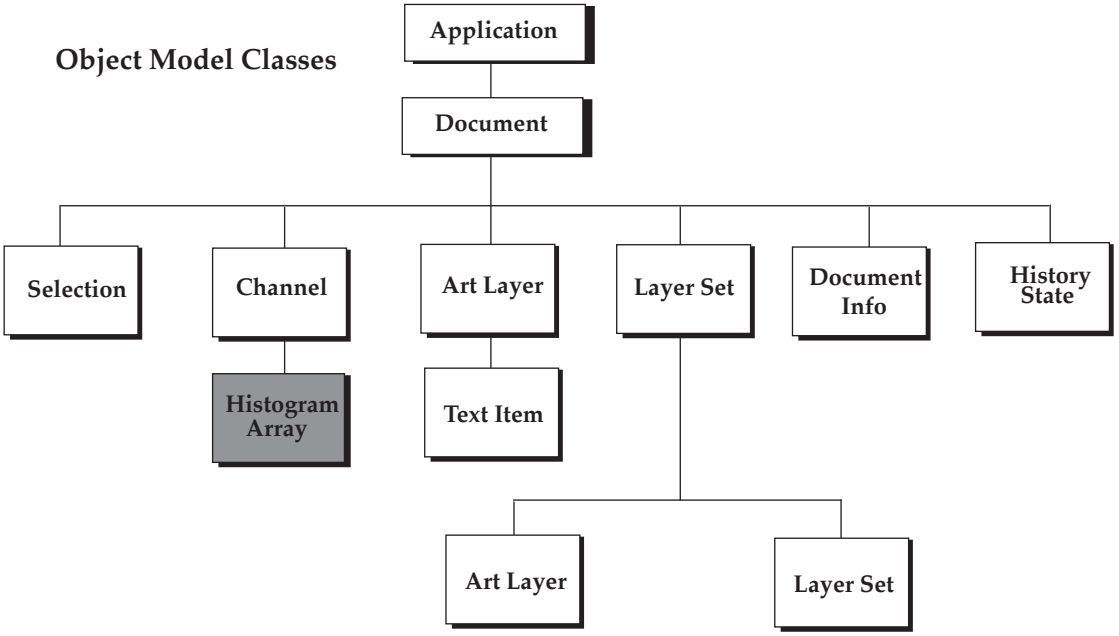
Whenever possible, JavaScript code samples are used to give real-world context to the topics under discussion. Many of these examples do not necessarily show the most efficient way to construct a JavaScript statement, but they are written to be easy to read and understand. Error checking code, for example, is brief in most of the examples—the point is to show you how to address and work with the Photoshop objects. Many of the examples may be combined to make scripts with greater functionality.

Photoshop's object model

As an aid to understanding how many of the most important classes available in Photoshop relate to each other, a brief description of the Object Model is given. A good understanding of Photoshop's object model will improve your scripting abilities.

In the object model illustrated below, the Photoshop Application object sits at the top of the containment hierarchy. The Document object, directly below the Photoshop application, is the active object you are working with and the gateway to the main components of the Photoshop object model.

The Document class is used to make modifications to the document image. By using the `Document` object you can crop, rotate or flip the canvas, resize the image or canvas, and trim the image. You could also use the `Document` object to get the active layer, for example, save the current document, then copy and paste within the active document or between different documents.



Selection Class

The Selection class is used to specify an area of pixels in the active document (or in a selected layer of the active document) that you want to work with.

Channel Class

The Channel class is used to store pixel information about an image's color. Image color determines the number of channels available. An RGB image, for example, has four default channels: one for each primary color and one for editing the image. You could have the red channel active in order to manipulate just the red pixels in the image, or you could choose to manipulate all the channels at once.

These kinds of channels are related to the document mode and are called "component channels. In addition to the component channels, Photoshop lets you to create additional channels. You can create a "spot color channel", a "masked area channel" and a "selected area channel."

Using the methods of a Channel object, you can create, delete and duplicate channels or retrieve a channel's histogram and change its kind or change the current channel selection.

Layer Classes

Photoshop has 2 types of layers: an `art layer` that can contain image contents and a `layer set` that can contain zero or more art layers.

An Art Layer is a layer class within a document that allows you to work on one element of an image without disturbing the others. Images are typically composed of multiple layers (see Layer Set, below). You can change the composition of an image by changing the order and attributes of the layers that comprise it.

A Text Item is a particular type of art layer that allows you to add type to an image. In Photoshop, a `text` item is implemented as a property of the art layer.

A Layer Set is a class that comprises multiple layers. Think of it as a folder on your desktop. Since folders can contain other folders, a layer set is recursive. That is, one layer set may call another layer set in the Object Model hierarchy.

History Class

The History class is a palette object that keeps track of changes made to a document. Each time you apply a change to an image, the new state of that image is added to the palette. These states are accessible from document object and can be used to reset the document to a previous state. A history state can also be used to fill a selection.

In AppleScript, if you create a document and then immediately try to get history state, Photoshop returns an error. You must first activate Photoshop -- make it the front-most application -- before you can access history states.

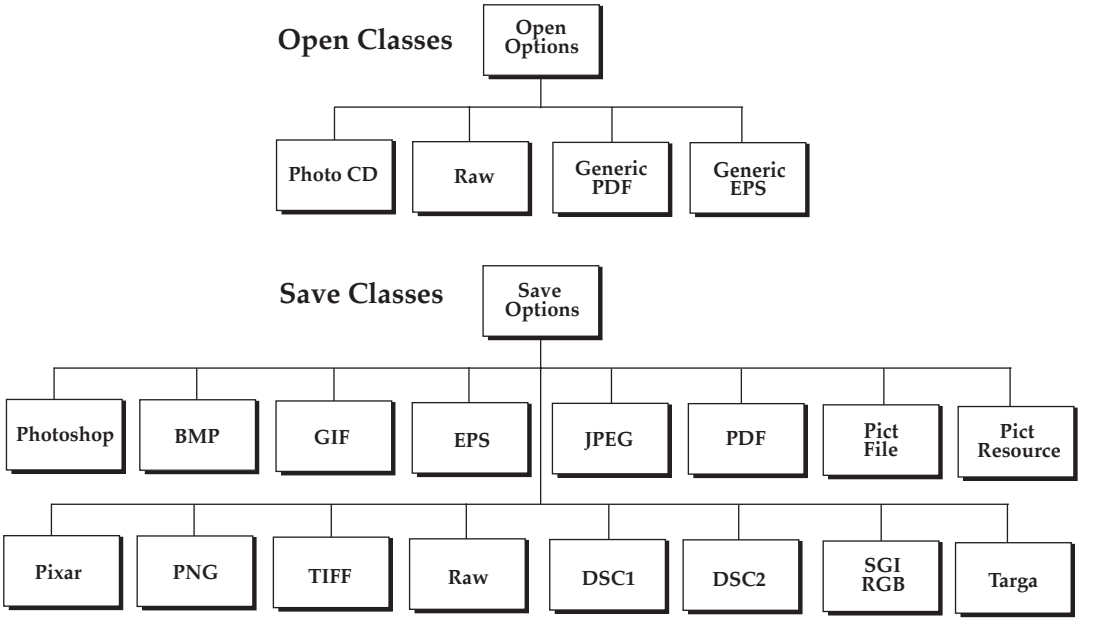
Document Info Class

The Document Info class stores metadata about a document. Metadata is any data that helps to describe the content or characteristics of a file.

Not shown in the Object Model are collections. A collection is a convenient way of grouping classes. Not all classes are associated with a collection.

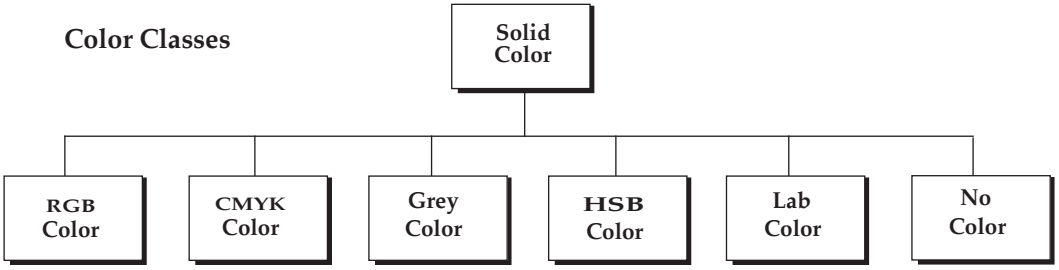
Additional Containment Classes

In addition to the classes described in the Object Model, other classes allow you to open and save objects in various formats and to specify color options.



Solid Color Classes

In Visual Basic and JavaScript, the `SolidColor` object handles all colors. The solid color classes available in Photoshop are illustrated below.



New in Photoshop CS

This section gives a brief overview of some important new JavaScript additions to the Photoshop application.

User Interface Elements

A JavaScript framework for creating User Interface (UI) elements is now included in Photoshop CS.

This framework allows developers to use JavaScript to create UI components such as windows, panels, buttons, checkboxes and so on. The framework -- called the **scripting user interface** -- is built as an abstraction layer on top of the windowing framework provided by the host platform on which Photoshop CS is running. Both Windows and MAC OS X native windowing systems are supported. For more information, see Chapter 2, Creating User Interface Elements.

File and Folder Methods

The following three new File and Folder methods allow users to interact with files using dialogs.

```
selectDialog (prompt, preset);
openDialog (prompt, select);
saveDialog (prompt, select);
```

For more information, see Chapter 3, Platform Interface.

User HOME directory (folder) shorthand character: "~"

You can now reference a file in a script that is stored in your home directory folder regardless of which platform the script is running on. For example:

```
var fileRef = new File("~/custdata.cfg");
```

Depending on the platform, `fileRef's` system-local path (`.fsName`) would look something like:

```
(Mac) "Mac OS 10.2:Users:username: custdata.cfg"
(Win) "C:\Documents and Settings\username\custdata.cfg"
(Unix) "/home/username/custdata.cfg"
```

For more information, see Chapter 3, Platform Interface.

New Preprocessing Statements

Several new commands have been added to the functionality available for JavaScript in a Photoshop context. These statements, specific to Adobe products, are not part of standard

JavaScript but will enhance your ability to take advantage of Photoshop features. These directives are embedded in JavaScript comments and are as follows:

//@includepath <pathSpecification>

The path or paths which the JavaScript interpreter will "walk" looking for a script specified in a //@include statement. If using //@includepath, the name of the script provided in the //@include statement should either be the name only without path qualifier or with a relative path. Multiple paths can be provided, separated by semi-colons. Given this:

```
//@includepath "~;../../FolderTwoLevelsUp"
//@include "jsLibrary.js"
//@include "libraries/otherLibrary.js"
```

The interpreter will search for jsLibrary.js in:

- The current working directory;
- The user's HOME directory ('~');
- The directory/folder 'FolderTwoLevelsUp' two levels up from the current working directory;

'otherLibrary.js' will be searched for in the same way, except that the directory/folder 'libraries' must be found in one of the search locations. The first match encountered will be loaded, so care must be taken when specifying the search order to ensure that the intended version of the file is found first.

//@include fileSpecification

The relative or fully qualified path/scriptname to include. The script is included inline into the active script, so care must be taken to ensure that variable names and function definitions in the included script do not collide with the active script. Inclusion occurs before the active script is evaluated. Note that lines numbers displayed in the debugger relate to the entire composite script, not the original line numbers of the active and included script – again, because the included file is not treated as a separate module but is rather included directly inline.

//@show include

If a script has been included in the active script, its code is not shown by default in the debugger. This statement

alert(), prompt(), confirm()

Although not strictly-speaking new, the following standard JavaScript built-in methods often slip by unnoticed by users. To compensate for this inadvertent oversight, they are presented here as a group. Please feel free to incorporate these methods into your scripts.

```
// Display information in a dialog to the user
alert( "Show me the money." );

// Prompt the user for input, providing a default value.
// Returns a string, or null if cancelled
var answer = prompt( "Enter an amount:", 100 );

// Ask the user a true/false question.
// OK=true, Cancel=false
var yesOrNo = confirm( "Proceed?" );
```

2

Creating User Interface Elements

A JavaScript framework for creating User Interface (UI) elements is included in Photoshop CS.

This framework allows developers to use JavaScript to create UI components such as windows, panels, buttons, checkboxes and so on. The framework -- called the **scripting user interface** -- is built as an abstraction layer on top of the windowing framework provided by the host platform on which Photoshop CS is running. Both Windows and MAC OS X native windowing systems are supported.

The motivation behind the creation of this scripting user interface was twofold:

- To enable JavaScripts to create dialogs and interact with controls. This satisfies a fundamental need on the part of developers to create parameterized scripts, whose actions can be controlled more directly by the end user.
- To extend the JavaScript environment to allow scripts to create UI elements dynamically. In this way, developers can create specialized interactive access to an application's functionality.

Types of Interface Elements

The following type of window is supported:

- `dialog` -- a modal dialog box. Photoshop CS supports **modal dialogs** only. Modeless dialogs, such as palettes, are **not** supported.

The following controls and UI elements are supported:

- Panels (frames) -- (classname `Panel`) a container to group and organize other control types
- Push buttons -- (classname `Button`) a button containing a text string
- Radio buttons-- (classname `RadioButton`) a dual-state control, usually grouped with other radio buttons, only one of which is set
- Checkbox buttons -- (classname `Checkbox`) a dual-state control showing a checked box (if true) or an empty box (if false)

- Edit text -- (classname `EditText`) an text field that the user **can** change.
- Static text -- (classname `StaticText`) a text field that the user **cannot** change
- Scrollbars -- (classname `Scrollbar`) a standard scrollbar with a moveable element and stepper buttons to incrementally move the element.
- Sliders -- (classname `Slider`) a standard slider with a moveable position indicator

In addition, the given classnames described above can used in window resource specifications to define controls within a window or panel. See “Creating a window using window resource specifications” on page 18 for more information.

JavaScript UI Interface

This section provides a description of the scripting user interface programming model.

UI Objects

The scripting user interface defines *Window* objects that wrap native windows and various control elements (Buttons, StaticText, etc.), which wrap simple native controls. These objects share common methods such as “query the element type”, “move the elements around”, and “set the title, caption or content”. For a complete list of properties and methods, see “JavaScript UI Reference” on page 30

Creating a window

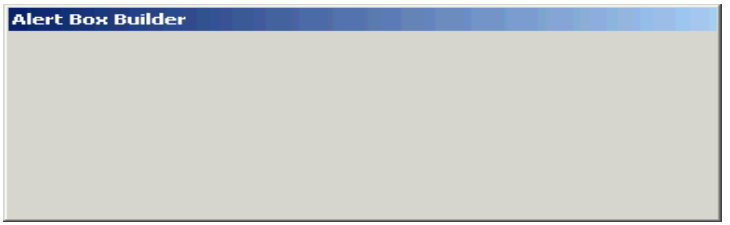
To create a new window, use the *Window* constructor function. The constructor takes the desired type of the window (`dialog`) as a parameter. You can supply optional arguments to specify an initial window title and bounds.

The code examples provided in the JavaScript Interface section consist of short segments from a complete script that is included later in this document. The examples presented build upon each other.

The following example creates an empty dialog with the variable name `dlg`. This dialog is carried though to subsequent examples:

```
// Create an empty dialog window near the upper left of the screen var
var dlg = new Window('dialog', 'Alert Box Builder', [100,100,480,245]);
dlg.show();
```

Note: Newly created windows are initially invisible; the *show()* method makes them visible.



Roughly speaking, the numeric parameters to the constructor correspond to the top left and bottom right coordinates of the window. The *bounds* supplied when creating the dialog specify the requested size of the client area, which is the area of the dialog on which you can create controls. It does not include the title bar and borders around the client area. The size and position of the dialog as a whole are automatically adjusted to maintain the requested client area size.

For a more detailed description of window *bounds*, see "Element size and location" on page 11.

Container elements

All windows are *containers*, which is to say they contain other elements such as panels, buttons and checkboxes within their boundaries.

Within a window, you can create other types of container elements and add interface components to them, just like you add elements to a window (see "Adding elements" on page 12). Elements added to a container are considered children of that container and certain operations performed on a container element also apply to its children. For instance, calling the container's *hide()* method makes the container invisible and makes all of its visible children invisible as well.

Along the same lines, calling the container's *show()* method makes the container visible as well as any child elements that were visible before the container was hidden. The following properties and methods of containers also apply to all children of that container: *visible*, *enabled*, *hide()*, *show()*.

Element size and location

To set the size and location of windows and controls, use the *bounds* property. As is typical when working with window systems, the location of a window is defined as the point (pair of coordinates) where the top left corner of the window is specified in the screen coordinate system.

The location of an element within a window or other container element is defined as the point where the top left corner of an element is specified in the window coordinate system, relative to the container the element lies within. Size is specified by width and height in pixels. A complete bounds specification therefore consists of 4 integer values which define the position of the upper left corner of the object and its dimensions.

The value of the *bounds* property can take several forms: a string with special contents, an inline JavaScript "Bounds" object, or a four-element array. The following examples show equivalent ways of placing a 380 by 390 pixel window near the upper left corner of the screen:

```
var dlg = new Window('dialog', 'Alert Box Builder', [100,100,480,490]);
dlg.bounds = [100,100,480,490];
dlg.bounds = {left:100, top:100, right:480, bottom:490};
dlg.bounds = "left:100, top:100, right:480, bottom:490";
```

Note that the window dimensions define the size of the "client area" of the window, which is the portion of the window that an application can directly control. The actual window size will typically be larger, because the host platform's window system can add title bars and borders to windows.

When read, the `bounds` property returns a *Bounds* object -- an array of four values representing the coordinates of the upper left and lower right corners of the element: *[left, top, right, bottom].*

Adding elements

To add elements to a window or other container, use the container's *add()* method. This method accepts the type of the element to be created and some optional parameters, depending on the element type. The return value is the UI object created or *null* on errors. The value of the element's visible property defaults to "true".

element is initially visible, but it will remain invisible as long as its parent object is invisible.

A second (optional) parameter may be used to specify the initial bounds. The bounds is relative to the working area of its parent container. For elements which display text, the text may be specified as the third (optional) parameter -- other types of elements have different semantics for a third argument.

For more information on the way in which each type of element interprets optional parameters, "JavaScript UI Reference" on page 30. These optional parameters are positional, meaning that if you want to specify some text for an element, but don't care about its bounds, you must still provide an argument for the second parameter in order to supply a value for the third (text) parameter. You can 'skip over' a positional parameter by specifying the '*undefined*' value as its argument value. In the example below, a Button element is created with an initial *text* value, but no *bounds* value.

```
dlg.btnPnl = dlg.add('panel', [15,330,365,375], 'Build it');
dlg.btnPnl.testBtn = dlg.btnPnl.add('button', undefined, 'Test');
```

Dynamically creating a property such as `btnPnl` to reference the control object returned by `add()` is not required, but can make it easier to later refer to the control. See "Accessing child elements" on page 13 for more information.

Creation properties

Some element types have attributes that may only -- in fact -- *can only* be specified when the element is created. These are not normal properties of the element, in that they cannot be changed during the element's lifetime, and they are only needed once. For these element types, an optional *creation properties* argument may be supplied to the *add()* method -- this argument is an object with one or more properties that controls things like the element's appearance, or special functions like 'read-only' for an edit text element.

All UI elements have a creation property called *name*, which can be used to assign a name for identifying that element. In the following example, the new *Button* element is assigned the name 'ok':

```
dlg.btnPnl.buildBtn = dlg.btnPnl.add('button', [125,15,225,35], 'Build',
       {name:'ok'});
```

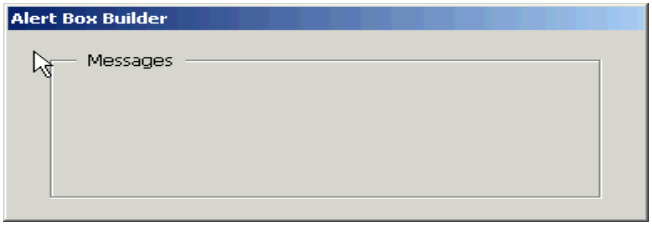
Accessing child elements

A reference to each element added to a window is appended to the window's *children* property.

The children collection is an array containing every defined element, indexed from 0 to the number of elements minus 1. For controls or other elements that do not have children, the *children* collection is empty.

The number of child elements in a window is equal to the value of the length property of the *children* collection. In the example below, since the '`msgPnl`' panel was the first element created in `dlg`, the text for the panel's title can be set as follows:

```
var dlg = new Window('dialog', 'Alert Box Builder',[100,100,480,245]);
dlg.msgPnl = dlg.add('panel', [25,15,355,130]);
dlg.children[0].text = 'Messages';
dlg.show();
```



Using creation properties, a name can be assigned to a newly created element. If this is done, a child can be referred to by its name. For instance, the *Button* in the example in the previous section was named 'ok', so the *Button* could now be referred to like this:

```
dlg.btnPnl.children['ok'].text = "Build";
```

An even simpler way to refer to a named child element is to use its name as a property of its parent element. We can also refer to the *Button* from the previous example like this:

```
dlg.btnPnl.ok.text = “Build”;
```

The value of an element’s internal *name* property is used by the scripting user interface when a script accesses a property of the element’s parent object that does not match any of the predefined properties.

In this case, the framework searches the *names* of the parent element’s children to see if a match exists, and if so, returns a reference to the matching child object.

Types of UI Elements

This section introduces the types of user interface elements you can create within a *Window* or other type of container element.

The Panel element

The *Panel* element is the only type of non-window container that is currently defined. *Panels* are typically used to visually organize related controls.

You can also use panels as separators: panels with width = 0 appear as vertical lines and panels with height = 0 appear as horizontal lines. When you create a Panel, you can specify an optional *borderStyle* property (used only at creation time) to control the appearance of the border drawn around the panel.

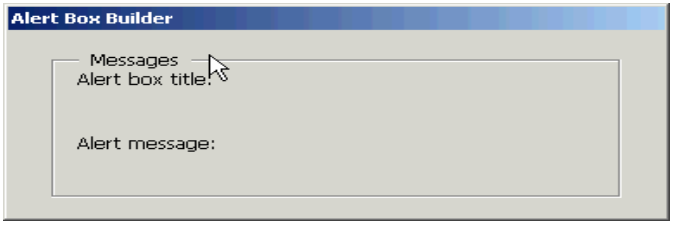
```
var dlg = new Window('dialog', 'Alert Box Builder',[100,100,480,245]);
dlg.msgPnl = dlg.add('panel', [25,15,355,130], 'Messages');
dlg.show();
```

The StaticText element

StaticText elements are typically used to display text strings that are not intended for direct manipulation by a user, like informative messages or identifying information for other elements. In the following example, a *Panel* is created, and several *StaticText* elements are added to it:

```
// sample code for section 2.6.2
var dlg = new Window('dialog', 'Alert Box Builder',[100,100,480,245]);
dlg.msgPnl = dlg.add('panel', [25,15,355,130], 'Messages');
dlg.msgPnl.titleSt = dlg.msgPnl.add('statictext', [15,15,105,35],
    'Alert box title:');
dlg.msgPnl.msgSt = dlg.msgPnl.add('statictext', [15,65,105,85],
    'Alert message:');
```

```
dlg.show();
```



The EditText element

EditText elements are typically used to provide a means for users to enter text to be supplied to the script when the dialog is dismissed. Text in *EditText* elements can be selected by a user and copied from or pasted into. The text property can be assigned to in order to display text in the element, and it can be read from to obtain the current text value.

The *textselection* property can be assigned to in order to replace the current selection with new text, or to insert text at the cursor (insertion point). It can be read from to obtain the current selection, if any.

Using the same panel pictured above, the example now adds some *EditText* elements, with initial values that a user can accept or replace:

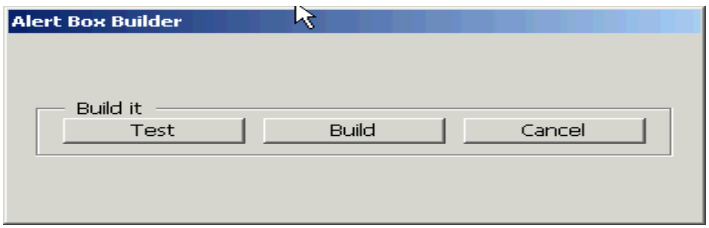
```
var dlg = new Window('dialog', 'Alert Box Builder',[100,100,480,245]);
dlg.msgPnl = dlg.add('panel', [25,15,355,130], 'Messages');
dlg.msgPnl.titleSt = dlg.msgPnl.add('statictext', [15,15,105,35],
     'Alert box title:');
dlg.msgPnl.titleEt = dlg.msgPnl.add('edittext', [115,15,315,35], 'Sample Alert');
dlg.msgPnl.msgSt = dlg.msgPnl.add('statictext', [15,65,105,85], 'Alert message:');
dlg.msgPnl.msgEt = dlg.msgPnl.add('edittext', [115,45,315,105],
    '<your message here>', {multiline:true});
dlg.show();
```

Note the *creation property* on the second *EditText* field, where *multiline:true* is specified. *multiline:true* indicates that the text in the item should wrap to the next page. In other words, it specifies a field in which a long text string may be entered, and the text will wrap to appear as multiple lines.

The Button element

Button elements are typically used to initiate some action from a *Window* when a user clicks the mouse pointer over the button; for example: accepting a dialog's current settings, canceling a dialog, bringing up a new dialog, etc. The text property provides a label to identify a *Button's* function:

```
var dlg = new Window('dialog', 'Alert Box Builder',[100,100,480,245]);
dlg.btnPnl = dlg.add('panel', [15,50,365,95], 'Build it');
dlg.btnPnl.testBtn = dlg.btnPnl.add('button', [15,15,115,35], 'Test');
dlg.btnPnl.buildBtn = dlg.btnPnl.add('button', [125,15,225,35],
     'Build', {name:'ok'});
dlg.btnPnl.cancelBtn = dlg.btnPnl.add('button', [235,15,335,35],
    'Cancel', {name:'cancel'});
dlg.show();
```



The Checkbox element

Checkbox elements are typically used to set the state of a Boolean variable in a script. A *Checkbox* is similar to a *Button*; it can be clicked by a user, and it has a *text* property to specify an identifying text string that appears next to the box.

When it is clicked, it changes its appearance, either showing a checkmark in the box area, or showing an empty box. When the checkmark appears, the state of the *value* property is true, and when the box is empty, the state of the *value* property is false. When you create a *Checkbox*, you can set its *value* property to specify its initial state and appearance.

```
//Add a checkbox to control the presence of buttons to dismiss the alert
//box
dlg.hasBtnsCb = dlg.add('checkbox', [125,145,255,165], 'Has
    alert buttons?');
dlg.hasBtnsCb.value = true;
```

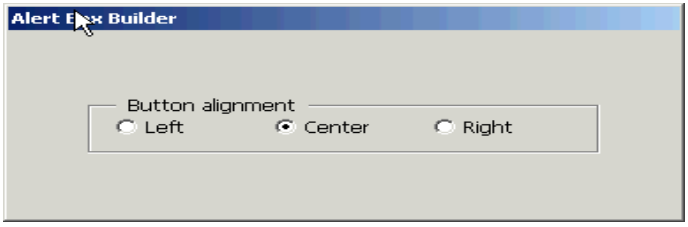
The RadioButton element

RadioButton elements are typically used to select one choice from 2 or more choices.

A *RadioButton* is similar to a *Button*; it can be clicked by a user, and it has a *text* property to specify an identifying text string that appears next to the button. Like a *Checkbox*, it has a *value* property that has a Boolean value, representing the state of that button.

You group a related set of *RadioButtons* by creating all the related elements one after another. The elements in a group interact with one another. Only one button's *value* can be true, and its appearance differs from others in the same group. Setting a different button's *value* property true changes the state of the button whose state was previously true to false. When you create a group of RadioButtons, you should set the state of one of them true:

```
var dlg = new Window('dialog', 'Alert Box Builder',[100,100,480,245]);
dlg.alertBtnsPnl = dlg.add('panel', [45,50,335,95], 'Button alignment');
dlg.alertBtnsPnl.alignLeftRb = dlg.alertBtnsPnl.add('radiobutton',
    [15,15,95,35], 'Left');
dlg.alertBtnsPnl.alignCenterRb = dlg.alertBtnsPnl.add('radiobutton',
     [105,15,185,35], 'Center');
dlg.alertBtnsPnl.alignRightRb = dlg.alertBtnsPnl.add('radiobutton',
     [195,15,275,35], 'Right');
dlg.alertBtnsPnl.alignCenterRb.value = true;
dlg.show();
```



The Slider element

Slider elements are typically used to select within a range of values, allowing the user to hold the mouse pointer down over a moveable position indicator on the slider and drag this indicator within the range of the slider. If you click the mouse pointer on a point on the slider bar, the position indicator will jump to that location.

A *Slider* has a *value* property that reflects the position of the moveable indicator, and *minvalue* and *maxvalue* properties to define the endpoints of the slider's range of values.

To make a slider control appear like those used in Photoshop, adjust the height of the control until the slider bar appears as a single line.

The Scrollbar element

Scrollbar elements are similar to *Slider* elements, in that they are often used to select within a range of values, and have a moveable position indicator. They have all the properties of sliders, and in

addition, they have 'stepper buttons' at each end of the scrollbar for moving the position indicator in fixed-size steps.

You can control the size of each 'step' by setting the *stepdelta* property. Clicking the mouse pointer ahead of or behind the position indicator makes the position indicator jump a fixed number of values toward the point where you clicked. You can control the size of this 'jump' by setting the *jumpdelta* property.

You can create scrollbars with horizontal or vertical orientation; if width is > height, the orientation is horizontal, otherwise it is vertical. The following example creates a *Scrollbar* element with associated *StaticText* and *EditText* elements within a panel:

```
dlg.sizePnl = dlg.add('panel', [60,240,320,315], 'Dimensions');
dlg.sizePnl.widthSt = dlg.sizePnl.add('statictext', [15,15,65,35],
    'Width:';
dlg.sizePnl.widthScrl = dlg.sizePnl.add('scrollbar',
    [75,15,195,35],300, 300, 800);
dlg.sizePnl.widthEt = dlg.sizePnl.add('edittext', [205,15,245,35]);
```

Note that the last 3 arguments to the *add*() method that creates the scrollbar define the values for the *value, minvalue* and *maxvalue* properties. *Scrollbars* are often created with an associated *EditText* field to display the current value of the scrollbar, and to allow setting the scrollbar's position to a specific value.

Creating a window using window resource specifications

A specially formatted string provides a simple and compact means of creating a window and its component elements as a *resource specification*. A resource specification allows you to define and configure multiple window components in one easy-to-reference script.

The special string is passed as the type parameter to the *Window* constructor function, as follows:

```
// create a new dialog from a resource specification
var alertBuilderResource =
    "dialog { text: 'Alert Box Builder', bounds:[100,100,480,490], \
        msgPnl: Panel { text: 'Messages', bounds:[25,15,355,130], \
            titleSt:StaticText { text:'Alert box title:', \
                bounds:[15,15,105,35] }, \
    titleEt:EditText { text:'Sample Alert', bounds:[115,15,315,35] }, \
     msgSt: StaticText { text:'Alert message:', \
        bounds:[15,65,105,85] }, \
     msgEt: EditText { text:'<your message here>', \
             bounds:[115,45,315,105], properties:{multiline:true} } \
}, \
hasBtnsCb: Checkbox { text:'Has alert buttons?', alignment:'center', \
                  bounds:[125,145,255,165] }, \
alertBtnsPnl: Panel { text:'Button alignment', bounds:[45,180,335,225], \
    alignLeftRb:RadioButton { text:'Left', bounds:[15,15,95,35] }, \
```

```
    alignCenterRb:RadioButton { text:'Center', \
        bounds:[105,15,185,35] }, \
    alignRightRb:RadioButton { text:'Right', bounds:[195,15,275,35] } \
  }, \
 sizePnl: Panel { text: 'Dimensions', bounds:[60,240,320,315], \
    widthSt:StaticText { text:'Width:', bounds:[15,15,65,35] }, \
    widthScrl:Scrollbar { minvalue:300, maxvalue:800, \
                bounds:[75,15,195,35] }, \
    widthEt:EditText { bounds:[205,15,245,35] }, \
    heightSt:StaticText { text:'Height:', bounds:[15,45,65,65] }, \
    heightScrl:Scrollbar { minvalue:200, maxvalue:600, \
                bounds:[75,45,195,65] }, \
    heightEt:EditText { bounds:[205,45,245,65] } \
     }, \
 btnPnl: Panel { text: 'Build it', bounds:[15,330,365,375], \
    testBtn:Button { text:'Test', bounds:[15,15,115,35] }, \
    buildBtn:Button { text:'Build', bounds:[125,15,225,35], \
                properties:{name:'ok'} }, \
    cancelBtn:Button { text:'Cancel', bounds:[235,15,335,35], \
                properties:{name:'cancel'} } \
  } \
 }";
 dlg = new Window (alertBuilderResource);
```

Note: This example creates the same dialog as the complete example script described in "JavaScript UI Example" on page 24, using a resource specification instead of explicit calls to the add() method of a container element.

The general structure of a window resource specification is a *Window* type specification (i.e., "dialog"), followed by a set of braces enclosing one or more property definitions. Controls are defined as properties within windows and other containers by specifying the classname of the control in a property definition, with properties of the control enclosed in braces {}, for example: `testBtn: Button { text: 'Test' }`.

Creation properties are specified in a *properties* property as named properties of an inline object (see example above). The syntax of window resource specification strings is completely described below.

Window resource specification syntax

The window resource specification syntax is given in BNF (Backus-Naur Form) below:

```
resourceSpec      = '"' windowTypeName inlineObject '"'
windowTypeName    = [a modal dialog]
inlineObject      = "{" propertiesList "}"
propertiesList    = propertyDefn { "," propertyDefn }
propertyDefn      = propertyName ":" propertyValue
propertyName      = [a JavaScript property name]
```

```
propertyValue     = "null" | "true" | "false" | string | number
                    | inlineArray |objectDefn
string            = [a JavaScript string literal]
number            = [any JavaScript integer or real number literal]
inlineArray       = "[" propertyValue { "," propertyValue } "]"
objectDefn        = ( namedObject | inlineObject )
namedObject       = [any object classname] inlineObject
```

Note: To create a UI element, the classname in the namedObject definition above can be any element classname referred to in "Types of Interface Elements" on page 9. For example:

```
"dialog { \
    text: 'From Resource', bounds: [10, 10, 210, 110], \
    box: Panel { \
        bounds: [10, 10, 190, 90], \
        ok: Button { \
            text: 'OK', bounds:[40, 30, 140, 50], \
        } \
    } \
}";
```

Interacting with controls: events and event callbacks

When a script creates a window, it typically adds control elements to the window that a user can manipulate, for instance, by clicking a button, entering text in an edittext field, moving a scrollbar, etc.

These user actions or manipulations generate *events* within the user interface system. The script that creates a window needs a way to be notified of events from that window or from controls within the window. The scripting user interface provides a number of *event callback methods* that a script can define as properties of any UI element that the script needs to interact with.

Each class of UI element has a set of callback methods defined for it. For windows, there are callbacks like *onClose(), onMove(), and onResize()*. For controls, callbacks vary from type to type. A typical callback is *onClick()* for button, radiobutton, and checkbox elements, and *onChange()* for edittext fields, scrollbars, and sliders.

To handle a given event for some UI element, simply define a property of the same name as the event callback in the element and assign a JavaScript function you have defined to it. The example below uses "in line" functions, which employ a unique syntax and do not require a name. However, you can also define the function elsewhere in the script. In that case, simply assign the name of the function to the event handler property. The scripting user interface calls these functions on event notifications if defined.

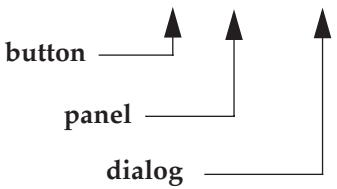
Examples:

```
/*'has buttons' checkbox enables or disables the panel that
determines the justification of the 'alert' button group */
dlg.hasBtnsCb.onClick =
    function () { this.parent.alertBtnsPnl.enabled = this.value; };

//The Build and Cancel buttons close this dialog
with (dlg.btnPnl) {
    buildBtn.onClick =
        function () { this.parent.parent.close(1); };
    cancelBtn.onClick =
        function () { this.parent.parent.close(2); };
};
```

Because event callback functions work as methods of the object in which they are defined, the functions have access to the object via the "this" JavaScript keyword. In the examples above, "this" refers to the UI object a given callback is defined in, so properties of the UI object can be accessed relative to the "this". For example, because each UI object has a *parent* property which is a reference to its container object, *this.parent* gets you a reference to the object's parent object.

To elaborate further on this point, a *button( )* is contained within a panel, which is contained within a window, all of which are ultimately closed. The progression is from smaller to larger UI moving from left to right.

```
buildBtn.onClick = function () {this.parent.parent.close(1);};
```



Also be aware that you can simulate user actions by sending an event notification directly to a UI element, via the element's *notify()* method. In this manner, a script can generate events in the controls of a window, as if a user was clicking buttons, entering text, moving a window, etc.

radiobutton and *checkbox* elements have a boolean *value* property; using *notify()* to simulate a click on these elements also changes the value of this property, just like clicking the element would do. For instance, if the *value* of a *checkbox* '*hasBtnsCb*' in our example above is true, the following example changes the value to false:

```
if (dlg.hasBtnsCb.value == true)
    dlg.hasBtnsCb.notify();
// dlg.hasBtnsCb.value is now false
```

A complete description of the different event callback methods and notify() can be found in the"Common Methods and Event Handlers" on page 35.

Modal dialogs

A modal dialog is initially invisible. When calling its *show()* method, the dialog is displayed and starts executing. The call to *show()* does not return until the dialog has been dismissed, typically by the user clicking an OK or Cancel button.

When calling the *hide()* or c*lose()* methods during the execution of a modal dialog, the dialog is dismissed. The *close()* method accepts an optional argument that the call to *show()* returns.

Warning: *You cannot use the JavaScript debugger to debug event callback functions for modal dialogs, because once the dialog starts executing, it captures all mouse events. Setting a breakpoint in an event callback function for a modal dialog will result in an apparent application hang if the breakpoint is ever reached.*

To work around this restriction, an effective debugging technique is to create your dialog, but not call its show() method to make it visible. Then use the debugger to call the notify() method on controls whose event callback functions you wish to debug. It's considered good design practice to keep the code in the event callback functions simple, while deferring the primary script logic execution until after the dialog has been dismissed.

Default and Cancel Elements

Modal dialogs can usually be dismissed by typing certain keyboard shortcuts. In addition to clicking the 'OK' or 'Cancel' buttons, typing the 'Enter' key normally produces the same results as clicking the 'OK' (or default) button, and typing the 'Esc' key is equivalent to clicking the 'Cancel' button. In each case, the keyboard shortcut is the same as if your script had called the *notify()* method for the associated *Button*. The dialog designer has explicit control over which *Button* elements are notified by these keyboard shortcuts: a newly-created dialog has *defaultElement* and *cancelElement* properties that are initially undefined. The dialog designer can set these properties to the objects representing the buttons that should be notified when the respective keyboard shortcut is typed.

The scripting user interface provides reasonable defaults if the *defaultElement* and *cancelElement* properties are still undefined when the dialog is about to be shown for the first time.

Default values for the *defaultElement* property are determined by the following algorithm:

- The scripting user interface searches the dialog's buttons for a button whose name property has the string value 'ok' (case is not important). If one is found, *defaultElement* is set to that object.
- If no matching named object is found, The scripting user interface searches the dialog's buttons for a button whose *text* property has the string value 'ok' (case is not important). If one is found, *defaultElement* is set to that object.

Default value for the *cancelElement* property are determined by the following algorithm:

- The scripting user interface searches the dialog's buttons for a button whose name property has the string value 'cancel' (case is not important). If one is found, *cancelElement* is set to that object.
- If no matching named object is found, the scripting user interface searches the dialog's buttons for a button whose *text* property has the string value 'cancel' (case is not important). If one is found, *cancelElement* is set to that object.

These algorithms handle most dialog boxes without the designer having to explicitly set these properties. When you add buttons to a dialog that will be used to dismiss the dialog, use *creation properties* to set the *name* property of such buttons to 'ok' or 'cancel', depending on the desired semantics; this precaution makes the above algorithm work properly even when the *text* of such buttons is localized. If the scripting user interface cannot find a matching button for either case, the respective property is set to *null*, which means that keyboard shortcuts for default or cancel will not notify any elements.

Guidelines for creating and using modal dialogs

When your script creates a dialog, you typically create controls that the user must interact with in order to enter values that your script will use. In general, you can minimize the number of event callback functions you attach to various controls in your dialogs, unless interaction with those controls changes the operation of the dialog itself. In most cases where you simply want to read the states of various controls when the dialog is dismissed, you do not need to handle events for them. For instance, you often don't need *onClick()* functions for every *checkbox* and *radiobutton* in your dialog: when the dialog is dismissed, read their states using their *value* properties.

Some exceptions to this guideline:

- *onChange()* functions are needed for *edittext* elements, if users enter values which must be validated (like a number within a range). The event callback must perform any necessary validation, and interact with the user on errors.
- Define *onClick()* for OK and Cancel buttons which close the dialog with a given value.

  Note: Perform this function only if you have **not** defined the *defaultElement* and/or *cancelElement* properties **or** named these buttons in such a way that they will automatically be identified as the "OK" and "Cancel" buttons.

Prompts and Alerts

Some JavaScript environments provide functions on the global window object to display message boxes or alert boxes and a prompt box that displays one or two lines of text and then allows the user to enter one line of text.

The scripting user interface defines functions *alert()*, *confirm()* and *prompt()* on the *Window* class that provides this standard functionality. The host application controls the appearance of these simple dialog boxes, so they are consistent with other alert and message boxes displayed by the application. See the "JavaScript UI Reference" on page 30 for details.

JavaScript UI Example

Having explored the individual scripting components that make up the user interface, you are now ready to see the parts assembled into real-world JavaScript code that produces a fully functional user interface.

The JavaScript UI code sample described below includes the following functions, which creates a simple user interface builder window populated with various panels, checkboxes, buttons and controls. When you run the builder, you can then cause it to create an Alert Box.

- *createBuilderDialog()* -- Creates an empty dialog window near the upper left of the screen and adds a title panel, a checkbox, a control panel and a panel with buttons to test parameters and create the Alert Box specification.
- *initializeBuilder()* --Sets up initial control states and attaches event callback functions to controls.
- *runBuilder()* -- Runs the builder dialog and returns the resulting Alert Box UI
- *createResource()* -- Creates and returns a string containing a dialog resource specification that creates the Alert Box UI using the parameters entered
- *stringProperty()* -- Returns a formatted string
- *arrayProperty()* -- Returns a formatted array
- *createTestDialog()* -- Creates a new Test dialog

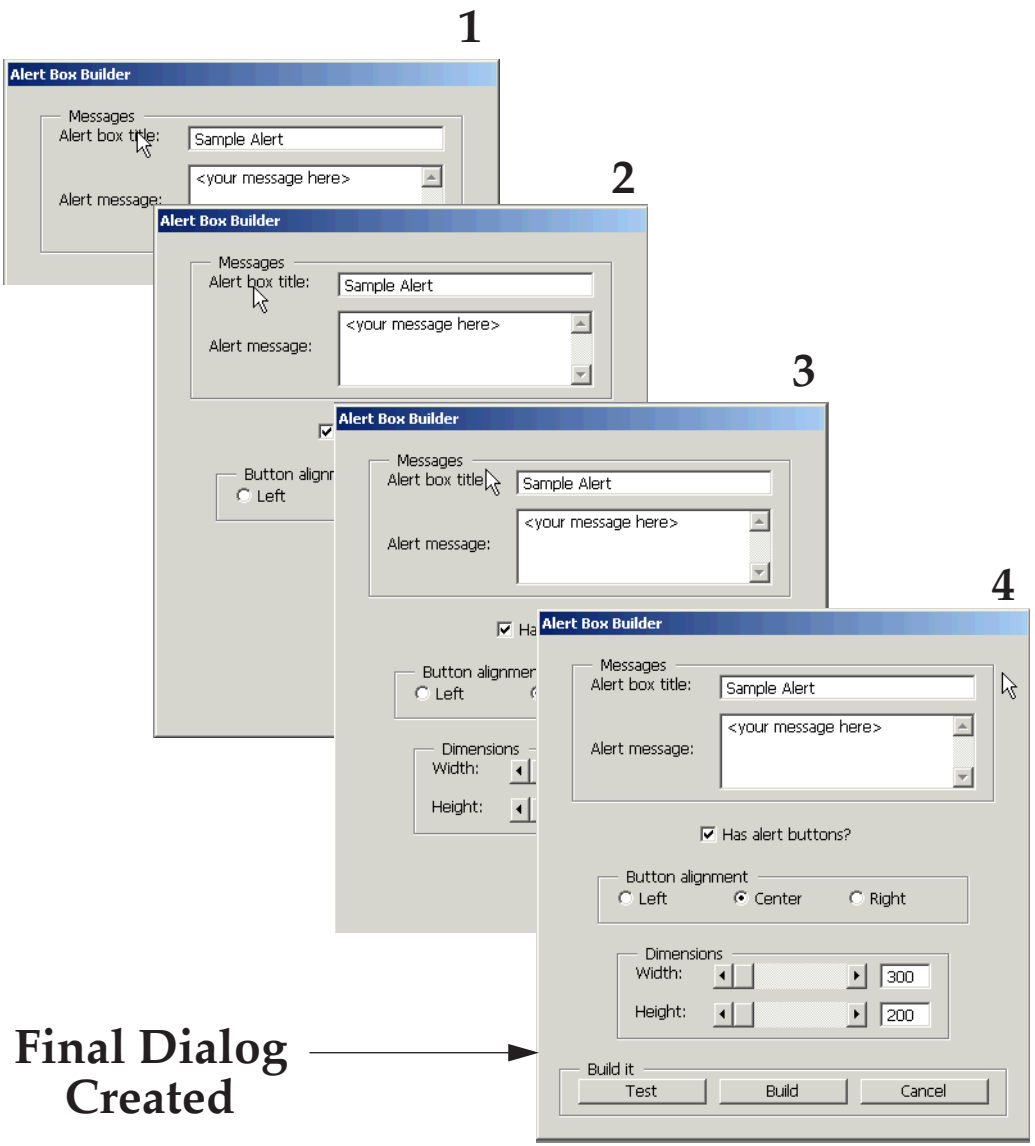
These functions are bundled together into a **Main** script, which assembles the final Alert Box dialog.

createBuilderDialog

Most of the heavy-lifting for visual components of the JavaScript UI code sample occurs in the *createBuilderDialog()* function, where the main components of the dialog are configured, as displayed below.

```
function createBuilderDialog()
{
    //Create an empty dialog window near the upper left of the screen
     var dlg = new Window('dialog', 'Alert Box Builder', [100,100,480,490]);                    1

    //Add a panel to hold title and 'message text' strings
     dlg.msgPnl = dlg.add('panel', [25,15,355,130], 'Messages');
     dlg.msgPnl.titleSt = dlg.msgPnl.add('statictext', [15,15,105,35], 'Alert box title:');
     dlg.msgPnl.titleEt = dlg.msgPnl.add('edittext', [115,15,315,35], 'Sample Alert');
     dlg.msgPnl.msgSt = dlg.msgPnl.add('statictext', [15,65,105,85], 'Alert message:');
     dlg.msgPnl.msgEt = dlg.msgPnl.add('edittext', [115,45,315,105], '<your message here>',
     {multiline:true});

    //Add a checkbox to control the presence of buttons to dismiss the alert box              2
     dlg.hasBtnsCb = dlg.add('checkbox', [125,145,255,165], 'Has alert buttons?');

    //Add panel to determine alignment of buttons on the alert box
     dlg.alertBtnsPnl = dlg.add('panel', [45,180,335,225], 'Button alignment');
     dlg.alertBtnsPnl.alignLeftRb = dlg.alertBtnsPnl.add('radiobutton', [15,15,95,35], 'Left');
     dlg.alertBtnsPnl.alignCenterRb = dlg.alertBtnsPnl.add('radiobutton', [105,15,185,35],
'Center');
     dlg.alertBtnsPnl.alignRightRb = dlg.alertBtnsPnl.add('radiobutton', [195,15,275,35], 'Right');

    //Add a panel with controls for the dimensions of the alert box
     dlg.sizePnl = dlg.add('panel', [60,240,320,315], 'Dimensions');
     dlg.sizePnl.widthSt = dlg.sizePnl.add('statictext', [15,15,65,35], 'Width:');
     dlg.sizePnl.widthScrl = dlg.sizePnl.add('scrollbar', [75,15,195,35], 300, 300, 800);      3
     dlg.sizePnl.widthEt = dlg.sizePnl.add('edittext', [205,15,245,35]);
     dlg.sizePnl.heightSt = dlg.sizePnl.add('statictext', [15,45,65,65], 'Height:');
     dlg.sizePnl.heightScrl = dlg.sizePnl.add('scrollbar', [75,45,195,65], 200, 200, 600);
     dlg.sizePnl.heightEt = dlg.sizePnl.add('edittext', [205,45,245,65]);

    //Add a panel with buttons to test parameters and create the alert box specification      4
     dlg.btnPnl = dlg.add('panel', [15,330,365,375], 'Build it');
     dlg.btnPnl.testBtn = dlg.btnPnl.add('button', [15,15,115,35], 'Test');
     dlg.btnPnl.buildBtn = dlg.btnPnl.add('button', [125,15,225,35], 'Build', {name:'ok'});
     dlg.btnPnl.cancelBtn = dlg.btnPnl.add('button', [235,15,335,35], 'Cancel', {name:'cancel'});

     return dlg;

} // createBuilderDialog
```

This code snippet, when broken down into smaller segments -- **and run in the context of the entire UI sample code that follows** -- produces the following succession of dialogs, which coalesce into one final Alert Box window.



For the final dialog to actually display, supporting code to initialize and run the Alert Box Builder must be included, as illustrated below.

```
function initializeBuilder(builder)
{
    //Set up initial control states
    with (builder) {
         hasBtnsCb.value = true;
         alertBtnsPnl.alignCenterRb.value = true;
         with (sizePnl) {
             widthEt.text = widthScrl.value;
             heightEt.text = heightScrl.value;
         }
    }

    //Attach event callback functions to controls

    /*'has buttons' checkbox enables or disables the panel that
         determines the justification of the 'alert' button group */

    builder.hasBtnsCb.onClick =
         function () { this.parent.alertBtnsPnl.enabled = this.value; };

    /*The edittext fields and scrollbars in sizePnl are connected */
    with (builder.sizePnl) {
         widthEt.onChange =
             function () { this.parent.widthScrl.value = this.text; };
         widthScrl.onChange =
             function () { this.parent.widthEt.text = this.value; };
         heightEt.onChange =
             function () { this.parent.heightScrl.value = this.text; };
         heightScrl.onChange =
             function () { this.parent.heightEt.text = this.value; };
    }

    with (builder.btnPnl) {
        //The Test button creates a trial Alert box from the current specifications
         testBtn.onClick =
             function () {
                 Window.alert('Type Enter or Esc to dismiss the test Alert box');
                 createTestDialog(createResource(this.parent.parent));
            };

        //The Build and Cancel buttons close this dialog
         buildBtn.onClick =
             function () { this.parent.parent.close(1); };
         cancelBtn.onClick =
             function () { this.parent.parent.close(2); };
   };
} // initializeBuilder

function runBuilder(builder)
{
    //Run the builder dialog, return its result
    return builder.show();
}

/*This function creates and returns a string containing a dialog
resource specification that will create an Alert dialog using
the parameters the user entered. */

function createResource(builder)
{
    //Define the initial part of the resource spec with dialog parameters
    var dlgWidth = Number(builder.sizePnl.widthEt.text);
    var dlgHeight = Number(builder.sizePnl.heightEt.text);
    var res = "dialog { " +
```

```
          stringProperty("text", builder.msgPnl.titleEt.text) +
          arrayProperty("bounds", 0, 0, dlgWidth, dlgHeight) +
            "\n";

     //Define the alert message statictext element, sizing it to the alert box
      var margin = 15; var l, t;
      var msgWidth, msgHeight;
      var hasButtons = builder.hasBtnsCb.value;
      var btnsHeightUsed = hasButtons ? 20 + margin : 0;
      msgHeight = 60;
      msgWidth = dlgWidth - (margin * 2);

      l = margin;
      t = (dlgHeight - msgHeight - btnsHeightUsed) / 2;
      res += "  msg: StaticText { " +
          stringProperty("text", builder.msgPnl.msgEt.text) +
          arrayProperty("bounds", l, t, l + msgWidth, t + msgHeight) +
          "justify:'center', properties:{multiline:true} }";

     //Define buttons if desired
      if (hasButtons) {
          var btnWidth = 90;
         //Align buttons as specified
          with (builder.alertBtnsPnl) {
              if (alignLeftRb.value)
                  l = margin;
              else if (alignCenterRb.value)
                  l = (dlgWidth - (btnWidth * 2 + 10)) / 2;
             else
                  l = dlgWidth - ((btnWidth * 2 + 10) + margin);
          }
          t = dlgHeight - btnsHeightUsed;
          res += ",\n" +
              "  okBtn: Button { " +
              stringProperty("text", "OK") +
              arrayProperty("bounds", l, t, l + btnWidth, t + 20) +
               "},\n";
          l += btnWidth + 10;
          res += "  cancelBtn: Button { " +
              stringProperty("text", "Cancel") +
              arrayProperty("bounds", l, t, l + btnWidth, t + 20) +
                "}";
      }

     //All done!
      res += "\n}";
      return res;
}

function stringProperty(pname, pval)
{
    return pname + ":'" + pval + "', ";
}

function arrayProperty(pname, l, t, r, b)
{
    return pname + ":[" + l + "," + t + "," + r + "," + b + "], ";
}

function createTestDialog(resource)
{
    var target = new Window (resource);
    return target.show();
}

//------------- Main script -------------//
var builder = createBuilderDialog();
initializeBuilder(builder);
```

```
if (runBuilder(builder) == 1) {
    //Create the Alert dialog resource specification string
     var resSpec = createResource(builder);
    //Write the resource specification string to a file, using the standard file open dialog
     var fname = File.openDialog('Save resource specification');
     var f = File(fname);
     if (f.open('w')) {
          var ok = f.write(resSpec);
           if (ok)
               ok = f.close();
          if (! ok)
               Window.alert("Error creating " + fname + ": " + f.error);
    }
}
```

Sample Code Summary

This sample code is used to demonstrate some practical applications of the scripting interface.. Here a few of the major intentions of the script:

- To provide a simple real-world example of creating a user interface with multiple components and controls
- To show how certain controls such as sliders and edit text boxes can interact
- To show how radio buttons work and how to set radio buttons to true and initialize them
- To show a multi-line text edit box as displayed in the messages panel of the dialog box
- To show how you can associate static text fields with edit text fields and static text with other types of controls
- To show how simple event callback functions work and how you can attach event handler functions to any controls that can generate events
- To show how to enable and disable sets of controls. For example, in the alert checkbox, if you unclick the checkbox then everything in the button alignment field suddenly gets greyed out.
- To demonstrate how you typically dismiss a modal dialog by providing an OK and Cancel button
- To show you can still read property values out of the dialog and its controls after the dialog has been dismissed

Resource Specification Sample Code

To run this JavaScript UI code using a resource specification, change the lines indicated below and include the resource specification sample code. For more information on resource specifications, refer to “Creating a window using window resource specifications” on page 18.

Note: This is a complete example of a resource specification string. The *alertBuilderResource()* code displayed below is a way to create the same main dialog box created by the *createBuilderDialog()* function.

```
//------------- Alternate dialog creation using resource specification -------------//
/*
To use this code, replace the line above that says
    var builder = createBuilderDialog();
  with
    var builder = createBuilderDialogFromResource();
*/

var alertBuilderResource =
    "dialog { text: 'Alert Box Builder', bounds:[100,100,480,490], \
        msgPnl: Panel { text: 'Messages', bounds:[25,15,355,130], \
            titleSt:StaticText { text:'Alert box title:', bounds:[15,15,105,35] }, \
            titleEt:EditText { text:'Sample Alert', bounds:[115,15,315,35] }, \
             msgSt: StaticText { text:'Alert message:', bounds:[15,65,105,85] }, \
             msgEt: EditText { text:'<your message here>', bounds:[115,45,315,105],
                properties:{multiline:true} } \
        }, \
        hasBtnsCb: Checkbox { text:'Has alert buttons?', alignment:'center',
            bounds:[125,145,255,165] }, \
        alertBtnsPnl: Panel { text:'Button alignment', bounds:[45,180,335,225], \
            alignLeftRb:RadioButton { text:'Left', bounds:[15,15,95,35] }, \
            alignCenterRb:RadioButton { text:'Center', bounds:[105,15,185,35] }, \
            alignRightRb:RadioButton { text:'Right', bounds:[195,15,275,35] } \
        }, \
        sizePnl: Panel { text: 'Dimensions', bounds:[60,240,320,315], \
            widthSt:StaticText { text:'Width:', bounds:[15,15,65,35] }, \
            widthScrl:Scrollbar { minvalue:300, maxvalue:800, bounds:[75,15,195,35] }, \
            widthEt:EditText { bounds:[205,15,245,35] }, \
            heightSt:StaticText { text:'Height:', bounds:[15,45,65,65] }, \
            heightScrl:Scrollbar { minvalue:200, maxvalue:600, bounds:[75,45,195,65] }, \
            heightEt:EditText { bounds:[205,45,245,65] } \
        }, \
        btnPnl: Panel { text: 'Build it', bounds:[15,330,365,375], \
          testBtn:Button { text:'Test', bounds:[15,15,115,35] }, \
         buildBtn:Button { text:'Build', bounds:[125,15,225,35], properties:{name:'ok'} }, \
         cancelBtn:Button { text:'Cancel', bounds:[235,15,335,35], properties:{name:'cancel'} } \
         } \
   }";

function createBuilderDialogFromResource()
{
    //Create from resource
     return new Window(alertBuilderResource);
} // createBuilderDialogFromResource
```

JavaScript UI Reference

The JavaScript user interface defines the global elements of the Window object and properties and methods of all the UI classes.

Global elements of the Window object

The following functions are class methods of the global *Window* class only; windows created via *new Window()* do not have these functions defined.

To call class methods, use the following example syntax: `Window.alert("Class method!");`

`alert (text)`

Displays the specified string in a user alert box that provides an OK button. The alert dialog is not intended for lengthy messages. When the string argument to the alert method is too long, the alert dialog truncates it.

`confirm (text)`

Displays the specified string in a self-sizing modal dialog box that provides Yes (default) and No buttons. When this user clicks one of these buttons, this method hides the dialog and returns a value indicating the button the user clicked to dismiss the dialog. A return value of *true* indicates that the user clicked the Yes button to dismiss the confirm box. The confirmation dialog displays lengthier messages than the alert and prompt dialogs do, but if this string is too long, the dialog truncates it.

`find (type, title)`

return value: *Object*

Finds an existing window already created by a script. *title* is the title of the window and type is modal dialog. This value is a hint in case more than one window has the same title; if the type is unimportant, null or an empty string can be passed. If the window was found, the corresponding JavaScript *Window* object is generated and returned; if the window cannot be determined, the return value is *null*.

`prompt (prompt [, default])`

Displays a modal dialog that returns the user's text input. When the dialog opens, it displays the given *prompt* text and its text edit field is initialized with any specified *default* text. When the user clicks OK to dismiss the dialog, it returns the text the user entered. If the user clicks the Cancel button in this dialog, this method's result is the value *null*.

Common object properties

The following table shows the common properties defined for each element type.

	Window	Panel	StaticText	EditText	Button	Checkbox	RadioButton	Scrollbar	Slider
`active`	x			x	x	x	x	x	x
`bounds`	x	x	x	x	x	x	x	x	x
`children`	x	x	x	x	x	x	x	x	x
`enabled`	x	x	x	x	x	x	x	x	x
`jumpdelta`								x	
`justify`	x	x	x	x	x	x	x		

	Window	Panel	StaticText	EditText	Button	Checkbox	RadioButton	Scrollbar	Slider
maxvalue								x	x
minvalue								x	x
parent	x	x	x	x	x	x	x	x	x
stepdelta								x	
text	x	x	x	x	x	x	x		
textselection				x					
type	x	x	x	x	x	x	x	x	x
value						x	x	x	x
visible	x	x	x	x	x	x	x	x	x

Properties

Following are the properties defined for each element types listed above.

Property	Type	Description
active	*Boolean*	Contains *true* if the object is active, *false* otherwise. An active floating dialog is the front-most dialog. A modal dialog that is visible is by definition the active dialog. An active control is the one which will accept keystrokes, or in the case of a Button, be activated (clicked) when the user types a return. Set this true to make a given control or dialog active.
bounds	*Bounds*	Contains a *Bounds* object describing the location and size of the element as array values representing the coordinates of the upper left and lower right corners of the element: [*left, top, right, bottom*]. These are screen coordinates for window elements, and window-relative coordinates for other elements. See "Element Size and Location " for a definition of the *Bounds* object.

Property	Type	Description
`children`	*Object*	The collection of UI elements that the UI object contains. This is an array indexed by number or by a string containing an element's name. The *length* property of this array is the number of child elements for container elements and is zero for controls; future implementations may return additional elements for composite controls. Read only.
`enabled`	*Boolean*	Contains true if the object is enabled, *false* otherwise. If set to true, control elements will accept input. If set to *false*, control elements will not accept input, and all types of elements may change to a 'grayed-out' appearance.
`jumpdelta`	*Number*	Contains the value to increment or decrement a *Scrollbar* element's position by, when the user clicks ahead or behind the moveable element of the *Scrollbar* to make the scroll position 'jump'.
`justify`	*String*	Controls justification of text in static text and edit text controls. The value is either *"left", "center"*, or "*right*" and the default value is left-justified. Some implementations may not fully support this property, and it may be ignored for some types of controls.
`maxvalue`	*Number*	Contains the maximum value that the *value* property can have. If *maxvalue* is reset less than *value*, *value* will be reset to *maxvalue*. If *maxvalue* is reset less than *minvalue*, *minvalue* will be reset to *maxvalue*.
`minvalue`	*Number*	Contains the minimum value that the *value* property can have. If *minvalue* is reset greater than *value*, *value* will be reset to *minvalue*. If *minvalue* is reset greater than *maxvalue*, *maxvalue* will be reset to *minvalue*.
`parent`	*Object*	The parent object of a UI object. This property returns *null* for window objects. Read only.
`placement`	*Bounds*	An alternate name for the *bounds* property; *bounds* is the preferred name, and use of *placement* is deprecated.
`stepdelta`	*Number*	Contains the value to increment or decrement a *Scrollbar* element's position by, when a stepper button at either end of the scrollbar is clicked.
`text`	*String*	The title, label or text. May be ignored for certain window types. For controls, its usage depends on the control type. Many controls like buttons use the text as a label, while other controls, such as edit fields, use the text to access its content.
`textselection`	*String*	Replace the current text selection with the specified text string, modifying the value of the *text* property. If there is no selection, the specified text is inserted into the *text* property string at the current insertion point. Reading the *textselection* property returns any selected text, or an empty string if there is no selection.

Property	Type	Description
`type`	*String*	Contains the type name of the element. For *Window* objects, this is the value of the first argument to the *Window* constructor function. For controls, this is the value of the first argument to the *add()* method. Read only.
`value`	*Boolean*	(for *Checkbox* and *RadioButton*) *true* if the control has been set (i.e., a checkbox shows a check mark), *false* if not set.
`value`	*Number*	(for *Scrollbar* and *Slider*) the value of the control, for instance, the position of the moveable part of a *Scrollbar* or *Slider*. If *value* is reset outside the bounded range *minvalue, maxvalue, value* is set to the closest boundary.
`visible`	*Boolean*	Contains *true* if the object is physically visible, *false* otherwise. If set to *false*, the UI object is hidden, and if set to *true*, the object is made visible.

Properties found only in Window elements

Window elements contain the following properties, in addition to those described in the previous section.

`defaultElement` -- *Object*

The element to notify when a user types the Enter key, with the intent to dismiss the dialog as if the "OK" button had been clicked.

`cancelElement` -- *Object*

The element to notify when a user types the Esc key (or the <*Cmd* .> combination on a Mac), with the intent to dismiss the dialog as if the "Cancel" button had been clicked.

Objects used as property values

The values of certain properties are represented by objects that the scripting interface defines. This section describes those objects. It includes a description of their semantics, ways to create them, and descriptions of their properties.

The Bounds Object

A *Bounds* object is used to define the boundaries of a *Window* or UI element within its coordinate space. You cannot directly create a *Bounds* object; one is created when you set an element's *bounds* property. Reading the *bounds* property always yields a *Bounds* object. *Bounds* contains an array describing the position and size of a UI element. The array values represent the coordinates of the upper left and lower right corners of the element: [*left, top, right, bottom*]. These are screen coordinates for window elements, and are relative to the coordinate space of the parent (container) element for other element types.

You can set an element's *bounds* property and indirectly create a *Bounds* object in any of these ways:

`e.bounds = Object`

The object must contain properties named *left, top, right, bottom,* or *x, y, width, height,* where each property has an integer coordinate value.

`e.bounds = Array`

The array must have integer coordinate values in the order [*left, top, right, bottom*].

`e.bounds = String`

The string must be an executable JavaScript inline object declaration, containing the same property names as in the object case just described.

See "Element size and location" on page 11 for examples.

A *Bounds* object may be accessed as an array. In addition, it supports the following properties

Property	Type	Description
`left`	*Number*	The 'x' coordinate value of the left edge of the element.
`top`	*Number*	The 'y' coordinate value of the top edge of the element.
`right`	*Number*	The 'x' coordinate value of the right edge of the element.
`bottom`	*Number*	The 'y coordinate value of the bottom edge of the element.
`x`	*Number*	Same as *left*.
`y`	*Number*	Same as *top*.
`width`	*Number*	*right - left*.
`height`	*Number*	*bottom - top*.

Common Methods and Event Handlers

Following are the common methods and event handlers defined for each element type.

	Window	Panel	StaticText	EditText	Button	Checkbox	RadioButton	Scrollbar	Slider
`add()`	x	x							
`center()`	x								
`close()`	x								
`hide()`	x	x	x	x	x	x	x	x	x
`notify()`				x	x	x	x	x	x

	Window	Panel	StaticText	EditText	Button	Checkbox	RadioButton	Scrollbar	Slider
`show()`	x	x	x	x	x	x	x	x	x
`onChange()`				x				x	x
`onClick()`					x	x	x		
`onClose()`	x								
`onMove()`	x								
`onResize()`	x								

Methods

Descriptions of the common methods and event handlers listed above follow:

Method	Returns	Description
`add (type [, bounds, text, { <creation properties> } ]);`	*Object*	Creates a new UI element and add it to the *children* array of its parent *Window* or *Panel* element. The optional parameter *bounds* is a *Bounds* object describing its position and size. This may also be a four-element array. The optional parameter *text* is assigned to the UI element as the initial text or title. The UI element itself decides how to use this string; it may be ignored. In general, a Button uses the text as its label, while a edit field uses it as its initial content. Internally, the text is assigned to the *text* property of the element. The optional parameter <*creation properties*> is an object with properties that specify attributes of the UI element that are used only when the element is created. <*creation properties*> are specific to the type of UI element, and are described below in the sections for each element type. The return value is the newly created UI element or *null* on errors.
`center([window])`	no return value	Centers a *Window* on screen, or optionally, within the specified window object.

Method	Returns	Description
`close ([value])`	no return value	Closes a *Window*. For modal dialogs, the optional value is returned as the result of the *show()* call that caused the dialog to display and execute.
`hide()`	no return value	Hides the element. If *hide()* is called on a modal dialog, dismiss the dialog and set the dialog result to 0. The application may choose to ignore this call for certain UI object types.
`notify([event])`	no return value	Sends a notification message to whatever listens to the UI object. *notify()* effectively lets you control a dialog programmatically. Calling this method with no argument on a control simulates the activation of the control; a Button signals that it has been clicked via its *onClick()* method, an EditText element tells its listener that it contents have changed via its *onChange()* method, and so on. You can supply an optional argument to *notify()*, which is the name of the event handler to call. For instance, to simulate a dialog *dlg* being moved by a user, you can send a notification message as follows: *dlg.notify("onMove")*.
`show()`	*Number*	Displays the UI object. A *Window* may choose to ignore the setting of the visibility state if it is not applicable, like for inspectors whose visibility is controlled by the application only. If *show()* is called for a modal dialog, the dialog is displayed and executed. The call to *show()* will not return until the dialog has been dismissed. The result of *show()* is the dialog result as supplied to *close()*. For all other elements, the result is 0.
`onClick()`	no return value	This method is called when a control has been activated by clicking it. Not all types of controls implement this callback. If you are interested in processing this event, define a function of this name in the control element.

Method	Returns	Description
`onChange()`	no return value	This method is called when the content of a control has been changed. Not all types of controls implement this callback. If you are interested in processing this event, define a function of this name in the control element.
`onClose()`	no return value	This method is called when a *Window* is closed. If you are interested in processing this event, define a function of this name in the *Window* object.
`onMove()`	no return value	This method is called when a *Window* has been moved. If you are interested in processing this event, define a function of this name in the *Window* object.
`onResize()`	no return value	This method is called when a *Window* has been resized. If you are interested in processing this event, define a function of this name in the *Window* object.

UI Object descriptions

This section describes UI objects such as windows, panels, buttons, checkboxes and so on.

Window object

To create a new *Window* object:

Method	Returns	Description
`new Window ("dialog" [, title, bounds]);`	*Object*	Creates a new *Window*. The required *type* argument contains the requested element type for a modal dialog. The optional *title* argument is used to set the window title, if specified. Optionally, a *Bounds* object or array may be supplied that describes the *bounds* of the window. If no bounds are given, a default bounds is chosen. The return value is the newly created window or *null* on errors.

The panel element

To add a *Panel* element to a window *w*:

Method	Returns	Description
`w.add ("panel" [, bounds, text, {<creation properties>} ]);`	*Object*	The optional parameter *bounds* defines the element's position and size. The optional parameter text is the *text* displayed in the border of the panel. The optional parameter *<creation properties>* is an object that can contain any of the following properties:

To add a *border style* around a panel.

Method	Returns	Description
`borderStyle`	*String*	Specifies the appearance of the border drawn around the panel. It can be one of: none, etched, raised, sunken, black. The default borderStyle is etched.

If you specify a *Panel* whose width is 0, it will appear as a vertical line; a *panel* whose height is 0 will appear as a horizontal line. Making a panel invisible will also hide all its children; making it visible again will also make visible those children that were visible when the panel was made invisible.

The statictext control

To add a *StaticText* element to a window *w*:

Method	Returns	Description
`w.add ("statictext" [, bounds, text, {<creation properties>}]);`	*Object*	The optional parameter *bounds* defines the element's position and size. The optional parameter text is the *text* displayed by the control. The optional parameter *<creation properties>* is an object containing any of the following properties:
`multiline`	*Boolean*	If false (default) the control accepts a single line of text. If true, the control accepts multiple lines, in which case the text wraps within the width of the control.
`scrolling`	*Boolean*	If false (default), the text displayed cannot be scrolled. If true, scrolling buttons appear and the text displayed can be vertically scrolled; this case implies *multiline*.

The edittext control

To add an *EditText* element to a window *w*:

Method	Returns	Description
`w.add ("edittext" [, bounds, text, {<creation properties>}]);`	*Object*	The optional parameter *bounds* defines the element's position and size. The optional parameter *text* is the initial text displayed by the control. The optional parameter *<creation properties>* is an object containing any of the following properties:
`multiline`	*Boolean*	If false (default) the control accepts a single line of text. If true, the control accepts multiple lines, in which case the text wraps within the width of the control.
`readonly`	*Boolean*	If false (default), the control accepts text input. If true, the control will not accept input text, but simply displays the contents of its text property.
`noecho`	*Boolean*	If false (default), the control displays text that is typed as input. If true, the control will not display input text (useful for password fields).

The *EditText* control calls the *onChange()* event method if the editable text is changed or if its *notify()* method is called. It also has a *textselection* property to access any text selection within the edit field.

The button control

To add a *Button* element to a window *w*:

Method	Returns	Description
`w.add ("button" [, bounds, text]);`	*Object*	The optional parameter *bounds* defines the element's position and size. The optional parameter *text* is the text displayed inside the button control.

The *Button* control calls the *onClick()* event method if the control is clicked or if its *notify()* method is called.

The checkbox control

To add a *Checkbox* element to a window *w*:

Method	Returns	Description
`w.add ("checkbox" [, bounds, text]);`	*Object*	The optional parameter *bounds* defines the element's position and size. The optional parameter *text* is the text displayed next to the checkbox control.

The *Checkbox* control calls the *onClick()* event method if the control is clicked or if its *notify()* method is called. It also has a *value* property which indicates whether the control is set or not.

The radiobutton control

To add a *RadioButton* element to a window *w*:

Method	Returns	Description
`w.add ("radiobutton" [, bounds, text]);`	*Object*	The optional parameter *bounds* defines the element's position and size. The optional parameter *text* is the text displayed next to the radiobutton control.

All *RadioButtons* in a group must be created sequentially, with no intervening creation of other element types. Only one *RadioButton* in a group can be set at a time; setting a different *RadioButton*

unsets the original one. The *RadioButton* control calls the *onClick()* event method if the control is clicked or if its *notify()* method is called. It also has a *value* property which indicates whether the control is set or not.

The scrollbar control

To add a *Scrollbar* element to a window *w*:

Method	Returns	Description
`w.add ("scrollbar" [, bounds, value, minvalue, maxvalue]);`	*Object*	The optional parameter *bounds* defines the element's position and size. The optional parameter *value* is the initial position of the moveable element. The optional parameters *minvalue* and *maxvalue* define the range of values that can be returned by changing the position of the moveable element.

The *Scrollbar* control will have a horizontal orientation if the specified width is greater than its height at creation time; its orientation will be vertical if its height is greater than its width. It calls the *onChange()* event method if the position of the moveable element is changed by the user, or if its *notify()* method is called. The value property contains the current position of the scrollbar's moveable position indicator within the scrolling area, within the range of *minvalue* and *maxvalue*.

The slider control

To add a *Slider* element to a window w:

Method	Returns	Description
`w.add ("slider" [, bounds, value, minvalue, maxvalue]);`	*Object*	The optional parameter *bounds* defines the element's position and size. The optional parameter *value* is the initial position of the moveable element. The optional parameters *minvalue* and *maxvalue* define the range of values that can be returned by changing the position of the moveable element.

All *Slider* controls have a horizontal orientation. The *Slider* control calls the *onChange()* event method if the position of the slider is changed by the user, or if its *notify()* method is called. The *value* property contains the current position of the slider's moveable position indicator, within the range of *minvalue* and *maxvalue*.

3

Platform Interface

Different platforms implement different types of file systems. As a result, notations for specifying files and folders differ dramatically from one file system to the next. File system drives are organized into folders (or directories) and folders typically contain files or other folders.

File systems are organized hierarchically and each file or folder hasd a position relative to the "top" of the file system. The complete description of a file's location in the file system is called a *path*.

File and Folder Objects

The File and Folder objects wrap the underlying file system. A File object corresponds to a disk file, while a Folder object matches a directory or folder.

To create a File or Folder object, use the corresponding *File()* and *Folder()* functions. You can also create them with the *new* operator if you like; both ways of calling *File()* or *Folder()* return a new object. The constructor accepts full or partial path names. In either case, the path stored internally is an absolute, full path name, so a File or Folder object, once created, always points to a fixed location of the disk. If you do not supply a file or folder name, a temporary name is generated. Path names are in a portable format (see below).

The Folder object supports file system functionality such as walking directories, creating, renaming or removing files, or resolving file aliases. The File class supports I/O functions to read or write files. Note that a File or Folder instance does not actually create a File or Folder in the file system, although each class has a method to accomplish this, if desired.

File and Folder objects can be used at any place where a path name is required; its conversion to a string (the *toString()* method) returns the name of the file or folder as an absolute path name in URI notation (see the *absoluteURI* property below). If you need the operating system specific file name, use the *fsName* property.

When you create two File objects that refer to the same disk file, they are treated as distinct objects. If you open one of the File objects for I/O, the operating system may inhibit access to the opened File object from the other File object because the disk file already is open.

There are several methods to distinguish between a File and a Folder object. Here are some examples:

```
if (f instanceof File) ...
if (typeof f.open == "undefined") ...//Folders have an open method
```

Path names

There are significant differences among Windows, Macintosh and Unix file systems. The File and Folder objects provide functionality that allows you to interact with these systems in both platform-specific and platform-neutral ways.

Absolute and relative path names

To maximize portability of path names, File and Folder objects accept platform-neutral as well as operating-system-specific path names. Both objects support an URI-like form of path names that is very close to Unix conventions. The format loosely follows the RFC 2396 specification. This chapter describes how these path names work.

You can use absolute path names and relative path names. Absolute path names start with one or two slash characters. These path names describe the full path from a root directory down to a file or folder. Relative path names start off a known location, the current directory. A relative path name starts either with a directory name or with one of the special names "." and "..". The name "." refercs to the current directory, and the name ".." refers to the parent directory. The slash character is used to separate path elements.

Path	Description
`/dir1/dir2/file`	An absolute path name, describing the file *file* in the directory *dir2*, which is in the directory *dir1*, which again is in the root directory.
`./file`	The file *file* in the current directory; you could also simple use *file* without the beginning "./" sequence.
`../file`	The file *file* in the parent of the current directory.
`../../file`	The file *file* in the grandparent of the current directory.
`../dir1/file`	The file *file* in the directory *dir1*, which is parallel to the current directory.

When using portable path names, it is always a good idea to use relative path names. Setting the current directory is as easy as assigning a new path name to the property *Folder.current*. Relative path names make you independent of different volume names on different machines and operating systems.

Internally, the File and Folder objects always operate on the operating system specific path names. You can always retrieve the "real" path name by looking at the *fsName* property.

Special characters -- characters that are not alphanumeric and not one of the characters / - - . ! ~ * ' ( ) -- are encoded in UTF-8 notation. The file "Däumling" therefore has the portable name "D%C3%A4umling". Along the same lines, "Macintosh HD" would become "Macintosh%20HD". This encoding scheme is compatible to RFC 2396 as well as to the global JavaScript functions *encodeURI()* and *decodeURI()*.

Volume names

There is no common location in the various file systems where the names of mounted volumes are stored. On Mac OS X, all mounted volumes are entries in the */Volumes* directory. On Unix systems, there is no convention at all, and Windows does not mount remote volumes at all; there are only drive letters.

For this reasons, the File and Folder objects support a common convention. A volume name may be the first part of an absolute path. The objects know where to look for the volume names on the Macintosh and Windows and they translate the volume names accordingly. On Unix, no translation takes place.

On some occasions, there may actually be a directory in the root folder that has the same name as the volume name. Imagine a folder "C:\C" on Windows. Since the File and Folder objects do not know whether "/c" would address the drive C: or the directory "C:\C", a path name where the first element designates both a volume name and a directory name always describes the directory name. If you really need to access the volume by name, you will have to use an operating system specific path specifier.

The home directory

A path name can also start with the tilde "~" character. This character stands for the user's home directory -- the syntax has been borrowed from Unix systems. The home directory is the user directory that the operating system assigns to you when you log in.

All three platforms (Windows, Unix and Mac OS X) provide a definition for a home directory based on the username of the current user, as illustrated below.

Note: Windows systems look for the environment variable HOME and use whatever directory is found there as the home directory. If the HOME environment variable is undefined, the system uses the user's home directory, typically located in the "Documents and Settings" folder, as the home directory. In a Windows system, therefore, the following path would *equate to* the HOME directory: `C:\Documents and Settings\<username>`. Also note that on Windows (as well as on Unix) you can override the default home directory assignment by creating a HOME environment variable. The path name stored in the HOME environment variable must be a Windows path name or a UNC path name, not a portable path name.

The following examples assume *jdoe* as the username, and "~/file" as the file.

Platform	Path	Description
Windows	`C:\home\file`	HOME set to C:\home
	`D:\file`	HOME set to D:\
	`C:\Documents and Settings\jdoe\file`	HOME not set
Mac OS X	`/Users/jdoe/file`	
Unix	`/home/jdoe/file`	This assumes that the HOME environment variable is set to /home/jdoe
	`/users/jdoe/file`	

Note: Path names may vary depending on the operating system version and language.

The following code is used to reference a file in a script that is stored in the user's home directory, regardless of the platform the script is running on.

```
var fileRef = new File( "~/custdata.cfg" );
```

Depending on the platform, the local path for the file reference (`.fsName`) would look something like this:

Mac

```
/Users/jdoe/custdata.cfg
```

Windows

```
C:\Documents and Settings\jdoe\custdata.cfg
```

Unix

```
/user/jdoe/custdata.cfg
```

Operating system specifics

There are a few operating system specific items that are discussed in this chapter. The last examples in each section show the usage of the home directory, assuming that the user logged in as *jdoe*.

Aliases on Windows and the Macintosh work much the same. All accesses to the alias file are transparently forwarded to the real file behind the alias file. Only the *rename()* and *remove()* calls affect an alias directly.

Windows

On Windows, volume names correspond to drive letters. The path `/c/temp/file`translates, therefore, to `C:\temp\file`. If the current drive would contain a root directory with the same name as a drive letter, the directory would take precedence over the drive letter. Assume there is a directory `D:\C`, and the current drive was D. In that case, the path `/c/temp/file` would translate to `D:\c\temp\file`. In order to access drive C, you would have to use the Windows path name conventions.

Both the slash and the backslash character are valid path element separators. The file system is not case sensitive.

To access remote volumes, use UNC path names of the form *//servername/sharename*. These path names are portable, because both Max OS X and Unix ignore multiple slash characters. Despite the name (the U in UNC stands for Universal), UNC names do not work for local volumes.

The following examples assume D is the current drive:

Portable Path Name	Windows Path Name
`/c/dir/file`	`c:\dir\file`
`/remote/dir/file`	`D:\remote\dir\file`
`/root/dir/file`	`D:\root\dir\file`
`~/dir/file`	`C:\Documents and Settings\jdoe\dir\file`

On Windows, all file system aliases (here called shortcuts) are actual files whose name end with the extension ".lnk". You should never use this extension directly, however. The File and Folder objects work fine without these extensions. Imagine a shortcut to the file *some.txt*. The Windows file name would actually be *some.txt.lnk*. Use *some.txt* to create a File object matching this link. The *alias* property of this object would return *true*, and the *resolve()* method of the object would return the File object of the actual file. This behavior is the same as on the Macintosh. The ".lnk" extension for Windows aliases is used transparently; that is, renaming a Windows shortcut file renames the file name portion and leaves the ".lnk" extension intact.

These behaviors are portable, but please keep in mind that Windows permits a file and its alias to reside in the same folder. If you have a file "test.txt" and its alias (which is "test.txt.lnk"), and you

create a File object with "test.txt" you will access the original file and not the alias file. There is no way for you to access the alias file on a Windows system in this case.

Macintosh

When Mac OS X boots, the boot volume is the root directory of the file system. All other volumes, including remote volumes, are part of the */Volumes* directory. When looking at the first element of a path name, the File and Folder objects do the following:

- If the name is the name of the boot volume, discard it.
- If the name is a volume name, prepend the path with */Volumes*.
- Otherwise, leave the path as is.

Mac OS X path names are not case sensitive, as opposed to all other Unix dialects.

The following examples assume the boot volume to be *MacOSX*, and a mounted volume *Remote*.

Portable Path Name	Mac OS X Path Name
`/macosx/dir/file`	`/dir/file`
`/remote/dir/file`	`/Volumes/remote/dir/file`
`/root/dir/file`	`/root/dir/file`
`~/dir/file`	`/Users/jdoe/dir/file`

Mac OS 9 is not longer supported as an operating system. The old notation of path names using the colon as a path separator, however, is still supported and translates as follows:

Mac OS 9 Path Name	Portable Path Name
`MacOSX:dir:file`	`/macosx/dir/file`
`Remote:dir:file`	`/remote/dir/file`
`Root:dir:file`	`/root/dir/file`

Unix

Since the path name conventions are closely modeled after the Unix model, no translation takes place. For the home directory, the HOME environment variable is used, which is part of any shell environment. Symbolic links are treated as file system aliases.

Unix path names are case sensitive.

Portable Path Name	Unix Path Name
`/macosx/dir/file`	`/macosx/dir/file`
`/remote/dir/file`	`/remote/dir/file`

Portable Path Name	Unix Path Name
`/root/dir/file`	`/root/dir/file`
`~/dir/file`	`/home/jdoe/dir/file`

Portability issues

If you have to use multiple different machines and operating systems, try to use relative path names, or try to originate your path names from the home directory. If impossible, work with Mac OS X and Unix aliases and UNC names on Windows, and store your files on a machine that is remote to your Windows machine so you can use UNC names.

For example, use the Unix machine *gonzo* as the data storage. If you set up an alias *share* in the root directory of *gonzo* and if you set up a Samba share at *share* pointing to the same data location, the path name `//gonzo/share/file` would work well for Windows, Macintosh and Unix machines.

Unicode I/O

Usually, the contents of a file are in some 8-bit encoding; most often, the current system encoding is used, like code page 1252 on Windows or Mac Roman on the Macintosh. When doing file I/O, the encoding used to convert between 8-bit character sets and Unicode is by default assumed to be the system encoding. You can, however, set a large number of encodings by setting the *encoding* property of a file to the name of the desired encoding. This name is one of the standard Internet names that are used to describe the encoding of HTML files. Typical examples are ASCII, X-SJIS, or ISO-8859-1. The File object attempts to find the corresponding encoder in the operating system. If present, this encoder will be used for subsequent I/O. Reading the *encoding* property returns the current encoding.

A special encoder, BINARY, is present for binary I/O. This encoder simply extends every 8-bit character it finds to a Unicode character between 0 and 255. When using this encoder to write binary files, the encoder writes the lower 8 bits of the Unicode character. If, for example, the Unicode character 1000 is written, the encoder actually writes the character 232 (1000 is 0x3E8, and 0xE8 gets written, which is 232).

The data of some of the common file formats (UCS-2, UCS-4, UTF-8, UTF-16) starts with a special Byte Order Mark (BOM) character ("\uFEFF"). The File method *open()* reads a few bytes of a file and tries to detect this character. If successful, the corresponding encoding is set automatically and the character is skipped. If there is no BOM character at the beginning of the file, open() reads the first 2 Kbytes of the file and checks whether the data might be valid UTF-8 encoded data, and sets the encoding to UTF-8 if so.

To write 16-bit Unicode files in UTF-16 format, use the encoding UCS-2. This encoding uses whatever endian format the host system supports. Make sure to write the Byte Order Mark character "\uFEFF" as the first character of the file. Do this also when using the UTF-8 encoding.

Error handling

Each object has an *error* property. If accessing a property or calling a method caused an error, this property contains a message describing the type of the error. On success, the property contains the empty string. The property can be set, but setting it only causes the error message to be cleared. If a file is open, assigning an arbitrary value to the property also resets its error flag.

Scriptable properties and methods

This section distinguishes among three different sets of methods.

- Constructors

  A constructor is a global function that is used to create the actual objects of a class. It has the same name as the class of the object that it creates (like File or Folder). When called, it returns a new instance of the desired object.

- Class methods

  These methods are attached to the constructor. They are used for working with objects that the constructor returns, but they do not require an actual object of that class to work upon. These methods are also often refereed to as static methods. A typical example is `File.openDialog()`, which returns a new File object if the user specifies a file in the Open dialog.

- Object methods

  These methods are attached to an instance of an object, because they need a specific object to act upon. The `open()` method of a File object is a typical example; in order to work correctly, it needs the path name of the disk file that the File object actually wraps.

  You may also see class properties and object properties referenced in a similar manner. Class properties provide access to general data related to a class, while object properties provide access to data specific to an object, like the creation date of a specific file.

To give you some idea of how these distinctions play out in practice, here are some real-world examples:

```
// File and Folder constructors are in fact class methods which return
// objects, as illustrated below:
var fRef = new File( "~/myDoc" );

//The Folder class method selectDialog() is called directly
var selectedFolder = Folder.selectDialog( "Pick a folder", "~/Documents" );
```

```
// This code illustrates a Property that returns the name of the current
// operating system
File.fs;

// To access one of the File class object properties, you need to
// use an object instance
fRef.exists;
```

A key point to remember is that class members **do not** require an instance of the class to be created in order to access them.

Common elements

Both the File and the Folder objects share a common set of properties and methods. All properties and methods resolve file system aliases automatically unless indicated otherwise.

Class properties

Property	Type	Description
`fs`	String	The name of the file system. Read-only. Possible values are "Windows", "Macintosh" or "Unix".

Class methods

Method	Returns	Description
`isEncodingAvailable (String name);`	Boolean	This method checks whether your system supports a specific encoding. You supply the name of the desired encoding; the method returns *true* if that encoding is available, *false* otherwise.
`decode (String what);`	String	The method *File.decode()* or *Folder.decode()* decodes its input string as required by RFC 2396. All characters with a numeric value greater than 127 are encoded in UTF-8; they are stored as escaped characters starting with the percent sign followed by two hex digits. Also the characters / - _ . ! ~ * ' ( ) are encoded the same way. The String " D%C3%A4umling" would be decoded as "Däumling".
`encode (String what);`	String	The method *File.encode()* or *Folder.encode()* encodes its input string as required by RFC 2396. All characters with a numeric value greater than 127 are encoded in UTF-8; they are returned as escaped characters starting with the percent sign followed by two hex digits. Also the characters / - _ . ! ~ * ' ( ) are encoded the same way. The string "Däumling" would be encoded as " D%C3%A4umling".

Object properties

Property	Type	Description
absoluteURI	String	The full path name for the object in URI notation. Read-only.
alias	Boolean	Returns *true* if the object refers to a file system alias. Read-only.
created	Date	The creation date of the object. If the object does not refer to a folder or file on disk, the value is *null*. Read-only.
error	String	Contains a message describing the last file system error. Setting this value clears any error message and resets the error bit for opened files.
exists	Boolean	Returns *true* if the path name of this object refers to an actually existing file or folder. Read only.
fsName	String	The file-system specific name of the object as a full path name. Read-only.
modified	Date	The date of the object's last modification. If the object does not refer to a folder or file on disk, the value is *null*. Read-only.
name	String	The name of the object without the path specification. Read-only.
parent	Folder	The folder object containing this object. If this object already is the root folder of a volume, the property value is *null*. Read-only.
path	String	The path portion of the absolute URI. If the name does not have a path, this property contains the empty string. Read-only.
relativeURI	String	The path name for the object in URI notation, relative to the current folder. Read-only.

Object methods

Method	Returns	Description
`execute();`	Boolean	Attempt to find the application associated with this file or folder. If found, load the application and cause it to load the file. This method may be used to execute a file much as it had been double-clicked in the Finder or Explorer. It can be used to run scripts, to launch other applications and much more. Folders pop open as if double-clicked. Since this method opens a severe security hole, it is disabled by default. The method returns immediately after launching the application. It does not wait for the application to terminate. It returns *true* if the launch was successful.
`getRelativeURI (String basePath);`	String	Calculate and return the relative URI, given a base path, in URI notation. If the base path is omitted, the path of the current folder is assumed.
`remove();`	Boolean	Delete the file or folder that this object represents. Folders must be empty before they can be deleted. The return value is *true* if the file or folder has been deleted. IMPORTANT: The *remove()* method deletes the referenced file or folder immediately. It does not move the referenced the file or folder to the system trash. The effects of the remove method cannot be undone. It is recommended that you prompt the user for permission to delete a file or folder before deleting it. The method does not resolve aliases; it rather deletes the file alias itself.

Method	Returns	Description
`rename (String newName);`	Boolean	Rename the object to the new name. The new name must not have a path. Returns true if the object was renamed. The method does not resolve aliases, but rather renames the alias file.
`resolve();`	File, Folder or null.	Attempt to resolve the file system alias that this object points to. If successful, a new File or Folder object is returned that points to the resolved file system element. If the object is not an alias, or if the alias could not be resolved, the return value is *null*.

The Folder object

A Folder object wraps the underlying file system and corresponds to a directory or folder.

Class properties

Property	Type	Description
`current`	Folder	The current folder is returned as a Folder object. Assigning either a Folder object or a string containing the new path name sets the current folder.
`startup`	Folder	The folder containing the executable image of the running application. Read-only.
`system`	Folder	The folder containing the operating system files. Read-only
`temp`	Folder	The default folder for temporary files. Read-only.
`trash`	Folder	The folder containing deleted items. Read-only

Constructor

```
Folder (path);
new Folder (path);
```

This function constructs a new Folder object. If the given path name refers to an already existing disk file, a File object is returned instead.

Parameter	Type	Description	Returns
`path`	String	The full or partial path name of the folder. The folder that the path name refers to does not need to exist. If he argument is omitted, a temporary name is generated.	Folder or File

Class Methods

```
selectDialog (prompt, preset);
```

Open a dialog box that permits you to select a folder using the OS specific folder select dialog. Both arguments are optional.

Parameter	Type	Description	Returns
`prompt`	String	The first argument displays a prompt text if the dialog allows the display of such a message.	If the user selected a folder and clicked the OK button, the return value is a Folder object pointing to the selected folder. If the user clicked the Cancel button instead, the return value is *null*.
`preset`	Folder	The second argument is a folder that is pre-selected when the dialog opens.	

For example: Class method `select.Dialog()`

The following code presents a dialog with which to interactively select a directory/folder. In this example, the dialog defaults to the local user HOME directory/folder as a starting location for browsing. The method returns a Folder object reference on success; null on failure.

```
var selectedFolder = Folder.selectDialog( “Select the inputfolder”, Folder( “~” ) );
```

Object methods

Method	Returns	Description
`create();`	Boolean	Attempt to create a folder at the location the path name points to. Returns *true* if the folder was created.
`getFiles (String mask);`	Array	Get a list of File and Folder objects contained in the folder object. The mask is the search mask for the file names. It may contain question marks and asterisks and is preset to * to find all files. Alternatively, a function may be supplied. This function is called with a File or Folder object for every file or folder in the directory search. If the function returns *true*, the object is added to the array. On Windows, all aliases end with the extension ".lnk". This extension is stripped from the file name when found to preserve compatibility with other operating systems. You can, however, search for all aliases by supplying the search mask "*.lnk". This is NOT recommended, however, because it is not portable. The return value is an array of File and/or Folder objects that correspond to the files found. The return value is *null* if the folder does not exist.

The File object

A File object wraps the underlying file system and corresponds to a disk file.

Constructor

```
File (path);
new File (path);
```

This function constructs a new File object. If the given path name refers to an already existing folder, a Folder object is returned instead. The CRLF sequence is preset to the system default, and the encoding is preset to the default system encoding.

Parameter	Type	Description	Returns
`path`	String	The full or partial path name of the folder. The folder that the path name refers to does not need to exist. If he argument is omitted, a temporary name is generated.	Folder or File

Class Methods

```
openDialog (prompt, select);
saveDialog (prompt, select);
```

Opens the built-in OS dialog to either select an existing file to open, or to select a file name to save a file into.

Parameter	Type	Description	Returns
`prompt`	String	An optional prompt that is displayed as part of the dialog if the dialog permits the display of an additional message.	Nothing
`select`	See Win and Mac versions below	This argument allows the pre-selection of the files that the dialog displays. Unfortunately, this argument is different on the Macintosh and on Windows, as described below.	
`select (Win)`	String	The Windows selection string is actually a list of file types with explanative text. This list is displayed in the bottom of the dialog as a drop-down list box so the user can select which types of files to display. The elements of this list are separated by commas. Each element starts with the descriptive text, followed by a colon and the file search masks for this text. Again, each search mask is separated by a semicolon. A Selection list that allowed the selection of all text files (*.TXT and *.DOC) or all files would look like this: `Text Files:*.TXT;*.DOC,All files:*` A single asterisk character is a placeholder for all files.	File Object
`select (Mac)`	Function	On the Macintosh, the optional second argument is a callback function. This function takes one argument, which is a File object. When the dialog is set up, it calls this callback function for each file that is about to be displayed. If the function returns anything else than *true*, the file is not displayed. This is only true fo the *openDialog()* method, the *saveDialog()* method ignores this callback method.	File Handler

For example: Class method `openDialog()`

The following code presents the user with a dialog with which to interactively select a file. The optional second argument, the form of which differs between Windows and Macintosh, provides a means to filter the list of files shown in the dialog display list. The point of the example is to show how to obtain a reference to an existing JavaScript script file. The method returns a Folder object reference on success; null on failure.

```
// Windows
if ( Folder.fs == “Windows” ) {
    var openFile = File.openDialog( “Select a JavaScriptfile”,
       “JavaScript files:*.js” );
} else if ( Folder.fs == “Macintosh” ) {
    var openFile = File.openDialog( “Select a JavaScriptfile”, fileFilter );}


// Mac only -- Accept entries which:
// have a ".js" extension (regardless of case);
// have a .type of 'TEXT';
// are Folders (which will allow browsing the Folder heirarchy)
function fileFilter( f )
{
    var jsExtension = “.js”;
    var lCaseName = f.name;
    lCaseName.toLowerCase();
    if ( lCaseName.indexOf(jsExtension ) == f.name.length
    – jsExtension.length )
       return true;
    else if ( f.type == “TEXT” )
       return true;
    else if ( f instanceof Folder )
       return true;
    else

       return false;
}
```

For example: Class method `saveDialog()`

The following code sample is similar to the one presented above for `openDialog()`. However, it is intended for use in specifying a file save target. Note that a name to a not-already-existing document can be entered by the user (including extension if desired), but that the dialog does not create to file, nor does it actually save a file if an existing file is selected. Rather, the reference

returned is used to perform that operation. When used in support of document operations in Photoshop, for example, that means that using the File object reference with the client's saveAs() method, would look something like this:

```
var saveRef = File.saveDialog( “Save As” );
if ( saveRef ) {
    app.activeDocument.saveAs( saveRef );}
else {
    alert( “Save cancelled” );
}
```

Object properties

Property	Type	Description
creator	String	The Macintosh file creator as a four-character string. On Windows, the return value is always "????". Read-only.
encoding	String	Gets or sets the encoding for subsequent read/write operations. The encoding is one of several predefined constants that follow the common Internet encoding names. Valid names are UCS-2, X-SJIS, ISO-8851-9, ASCII or the like. A special encoder, BINARY, is used to read binary files. This encoder stores each byte of the file as one Unicode character regardless of any encoding. When writing, the lower byte of each Unicode character is treated as a single byte to write. See appendix A for a list of encodings. If an unrecognized encoding is used, the encoding reverts to the system default encoding.
eof	Boolean	This property has the value *true* if a read attempt caused the current position to be behind the end of the file. Read only. If the file is not open, the value is *true.*
hidden	Boolean	Returns *true* if the file is invisible. Assigning a Boolean value sets or clears this attribute.
length	Number	The size of the file in bytes. When setting the file size, the file must not be open.
lineFeed	String	The way line feed characters are written. This can be one of the three values *macintosh*, *unix* or *windows* (actually, only the first character is interpreted).
readonly	Boolean	This attribute, when set, prevents the file from being altered or deleted.
type	String	The Macintosh file type as a four-character string. On the Macintosh, the file type is returned. On Windows, "appl" is returned for .EXE files, "shlb" for .DLLs and "TEXT" for any other file. If the file does not exist, the file type is "????". Read-only.

Object methods

`close();`

Closes the open file. The return value is true if the file was closed, *false* on I/O errors.

Parameter	Type	Description	Returns
none			Boolean

`copy (target);`

Copies the file to the given location. You can supply an URI path name as well as another File object. If there is a file at the target location, it is overwritten. The method returns *true* if the copy was successful, *false* otherwise. The method resolves any aliases to find the source file.

Parameter	Type	Description	Returns
`target`	String/File	The target location.	Boolean

`open (mode, type, creator);`

Open the file for subsequent read/write operations. The *type* and *creator* arguments optional arguments that are Macintosh specific; they specify the file type and creator as two four-character strings. They are used if the file is newly created. On other platforms, they are ignored.

When *open()* is used to open a file for read access, the method attempts to detect the encoding of the open file. It reads a few bytes at the current location and tries to detect the Byte Order Mark character 0xFFFE. If found, the current position is advanced behind the detected character and the *encoding* property is set to one of the strings UCS-2BE, UCS-2LE, UCS4-BE, UCS-4LE or UTF-8. If the marker character cannot be found, it checks for zero bytes at the current location and makes an assumption about one of the above formats (except for UTF-8). If everything fails, the *encoding* property is set to the system encoding. The method resolves any aliases to find the file.

You should be careful if you try to open a file more than once. The operating system usually permits you to do so, but if you start writing to the file using two different File objects, you may destroy your data!

The return value is *true* if the file has been opened successfully, *false* otherwise.

Parameter	Type	Description	Returns
mode	String	**r (read)** Opens for reading. If the file does not exist or cannot be found the call fails. **w (write)** Opens an empty file for writing. If the file exists, its contents are destroyed. **e (edit)** Opens an existing file for reading and writing.	Boolean
type	String	The Macintosh file type; a four-byte character string; ignored on non-Macintosh operating systems.	
creator	String	The Macintosh file creator; a four-byte character string; ignored on non-Macintosh operating systems.	

```
read (chars);
```

Read the contents of the file from the current position on. Returns a string that contains up to the number of characters that were supposed to be read.

Parameter	Type	Description	Returns
chars	Number	The number of characters to read. If the number of characters to read is not supplied, the entire file is read in one big chunk, starting at the current position. If the file is encoded, multiple bytes may be read to create single Unicode characters.	String

```
readch();
```

Read one single text character. Line feeds are recognized as CR, LF, CRLF or LFCR pairs. If the file is encoded, multiple bytes may be read to create single Unicode characters.

Parameter	Type	Description	Returns
none			String

`readln();`

Read one line of text. Line feeds are recognized as CR, LF, CRLF or LFCR pairs. If the file is encoded, multiple bytes may be read to create single Unicode characters.

Parameter	Type	Description	Returns
none			String

`seek (pos, mode);`

Seek to a certain position in the file. Returns *true* if the position was changed. This method does not permit seeking to positions less than 0 or greater than the current file size.

Parameter	Type	Description	Returns
pos	Number	The new current position inside the file as an offset in bytes, dependent on the seek mode.	Boolean
mode	Number	The seek mode (0 = seek to absolute position, 1 = seek relative to the current position, 2 = seek backwards from the end of the file).	

`tell();`

Returns the current position in the file as a an offset in bytes.

Parameter	Type	Description	Returns
none			Number

`write (text, …);`

Write the given string to the file. The parameters of this function are concatenated to a single string. Returns *true* on success. For encoded files, writing a single Unicode character may result in multiple bytes being written. You should take care not to write to a file that is open in another application or object. This may cause loss of data, since a second write issued from another File object may overwrite your data!

Parameter	Type	Description	Returns
text	String	All arguments are concatenated to form the string to be written.	Boolean

```
writeln (text, …);
```

Write the given string to the file. The parameters of this function are concatenated to a single string, and a Line Feed sequence is appended. Returns true on success. If the file is encoded, multiple bytes may be read to create single Unicode characters.

Parameter	Type	Description	Returns
`text`	String	All arguments are concatenated to form the string to be written.	Boolean

Error messages

The following messages may be returned in the *error* property.

Error Message	Description
File does not exist	The file or folder does not exist, but the parent folder exists.
File exists	The file or folder already exists.
File is not open	An I/O operation was attempted on a file that was closed.
EOF	Attempt to read behind the end of a file.
Bad encoding	Unable to read encoded characters from a file.
Permission denied	The OS did not allow the attempted operation.
Cannot change directory	Cannot change the current folder.
Cannot create	Cannot create a folder.
Cannot rename	Cannot rename a file or folder.
Cannot delete	Cannot delete a file or folder.
I/O error	Unspecified I/O error.
Cannot set size	Setting the file size failed.
Cannot open	Opening of a file failed.
Cannot close	Closing a file failed.
Read error	Reading from a file failed.
Write error	Writing to a file failed.
Cannot seek	Seek failure.
Cannot execute	Unable to execute the specified file.

Supported encoding names

The following list of names is a basic set of encoding names supported by the File object. Some of the character encoders are built in, while the operating system is queried for most of the other encoders. Depending on the language packs installed, some of the encodings may not be available. Names that refer to the same encoding are listed in one line. Underlines are replaced with dashes before matching an encoding name.

Note, however, that the File object cannot process extended Unicode characters with values greater than 65535. These characters are left encoded as specified in the UTF-16 standard in as two characters in the range from 0xD700-0xDFFF.

Built-in encodings are:

```
US-ASCII, ASCII,ISO646-US,I SO-646.IRV:1991, ISO-IR-6,
ANSI-X3.4-1968,CP367,IBM367,US,ISO646.1991-IRV
UCS-2,UCS2, ISO-10646-UCS-2
UCS2LE,UCS-2LE,ISO-10646-UCS-2LE
UCS2BE,UCS-2BE,ISO-10646-UCS-2BE
UCS-4,UCS4, ISO-10646-UCS-4
UCS4LE,UCS-4LE,ISO-10646-UCS-4LE
UCS4BE,UCS-4BE,ISO-10646-UCS-4BE
UTF-8,UTF8,UNICODE-1-1-UTF-8,UNICODE-2-0-UTF-8,X-UNICODE-2-0-UTF-8
UTF16,UTF-16,ISO-10646-UTF-16
UTF16LE,UTF-16LE,ISO-10646-UTF-16LE
UTF16BE,UTF-16BE,ISO-10646-UTF-16BE
CP1252,WINDOWS-1252,MS-ANSI
ISO-8859-1,ISO-8859-1,ISO-8859-1:1987,ISO-IR-100,LATIN1
MACINTOSH,X-MAC-ROMAN
BINARY
```

The ASCII encoder raises errors for characters greater than 127, and the BINARY encoder simply converts between bytes and Unicode characters by using the lower 8 bits. The latter encoder is convenient for reading and writing binary data.

Additional encodings

In Windows, all encodings use so-called code pages. These code pages are assigned numeric values. The usual Western character set that Windows uses is e.g. the code page 1252. Windows code pages may be selected by prepending the number of the code page with "CP" or "WINDOWS- like "CP1252" for the code page 1252. The File object has a lot of other encoding names built-in that match predefined code page numbers. If a code page is not present, the encoding cannot be selected.

On the Macintosh, encoders may be selected by name rather than by code page number. The File object queries the Macintosh OS directly for an encoder. As far as Macintosh character sets are identical with Windows code pages, the Macintosh also knows the Windows code page numbers.

On Unix systems, the number of available encoders depends on the installation of the *iconv* library.

Common encoding names

The following encoding names are implemented both on Windows and Macintosh systems:

```
UTF-7,UTF7,UNICODE-1-1-UTF-7,X-UNICODE-2-0-UTF-7
ISO-8859-2,ISO-8859-2,ISO-8859-2:1987,ISO-IR-101,LATIN2
ISO-8859-3,ISO-8859-3,ISO-8859-3:1988,ISO-IR-109,LATIN3
ISO-8859-4,ISO-8859-4,ISO-8859-4:1988,ISO-IR-110,LATIN4,BALTIC
ISO-8859-5,ISO-8859-5,ISO-8859-5:1988,ISO-IR-144,CYRILLIC
ISO-8859-6,ISO-8859-6,ISO-8859-6:1987,ISO-IR-127,ECMA-114,ASMO-708,ARABIC
ISO-8859-7,ISO-8859-7,ISO-8859-7:1987,ISO-IR-126,ECMA-118,ELOT-928,GREEK8,GREEK
ISO-8859-8,ISO-8859-8,ISO-8859-8:1988,ISO-IR-138,HEBREW
ISO-8859-9,ISO-8859-9,ISO-8859-9:1989,ISO-IR-148,LATIN5,TURKISH
ISO-8859-10,ISO-8859-10,ISO-8859-10:1992,ISO-IR-157,LATIN6
ISO-8859-13,ISO-8859-13,ISO-IR-179,LATIN7
ISO-8859-14,ISO-8859-14,ISO-8859-14,ISO-8859-14:1998,ISO-IR-199,LATIN8
ISO-8859-15,ISO-8859-15,ISO-8859-15:1998,ISO-IR-203
ISO-8859-16,ISO-885,ISO-885,MS-EE
CP850,WINDOWS-850,IBM850
CP866,WINDOWS-866,IBM866
CP932,WINDOWS-932,SJIS,SHIFT-JIS,X-SJIS,X-MS-SJIS,MS-SJIS,MS-KANJI
CP936,WINDOWS-936,GBK,WINDOWS-936,GB2312,GB-2312-80,ISO-IR-58,CHINESE
CP949,WINDOWS-949,UHC,KSC-5601,KS-C-5601-1987,KS-C-5601-1989,ISO-IR-149,KOREAN
CP950,WINDOWS-950,BIG5,BIG-5,BIG-FIVE,BIGFIVE,CN-BIG5,X-X-BIG5
CP1251,WINDOWS-1251,MS-CYRL
CP1252,WINDOWS-1252,MS-ANSI
CP1253,WINDOWS-1253,MS-GREEK
CP1254,WINDOWS-1254,MS-TURK
CP1255,WINDOWS-1255,MS-HEBR
CP1256,WINDOWS-1256,MS-ARAB
CP1257,WINDOWS-1257,WINBALTRIM
CP1258,WINDOWS-1258
CP1361,WINDOWS-1361,JOHAB
EUC-JP,EUCJP,X-EUC-JP
EUC-KR,EUCKR,X-EUC-KR
HZ,HZ-GB-2312
X-MAC-JAPANESE
X-MAC-GREEK
X-MAC-CYRILLIC
X-MAC-LATIN
X-MAC-ICELANDIC
X-MAC-TURKISH
```

Additional Windows encoding names

```
CP437,IBM850,WINDOWS-437
CP709,WINDOWS-709,ASMO-449,BCONV4
EBCDIC
KOI-8R
KOI-8U
ISO-2022-JP
ISO-2022-KR
```

Additional Macintosh encoding names

These names are alias names for encodings that the Macintosh operating system might know.

```
TIS-620,TIS620,TIS620-0,TIS620.2529-1,TIS620.2533-0,TIS620.2533-1,ISO-IR-166
CP874,WINDOWS-874
JP,JIS-C6220-1969-RO,ISO646-JP,ISO-IR-14
JIS-X0201,JISX0201-1976,X0201
JIS-X0208,JIS-X0208-1983,JIS-X0208-1990,JIS0208,X0208,ISO-IR-87
JIS-X0212,JIS-X0212.1990-0,JIS-X0212-1990,X0212,ISO-IR-159
CN,GB-1988-80,ISO646-CN,ISO-IR-57
ISO-IR-16,CN-GB-ISOIR165
KSC-5601,KS-C-5601-1987,KS-C-5601-1989,ISO-IR-149
EUC-CN,EUCCN,GB2312,CN-GB
EUC-TW,EUCTW,X-EUC-TW
```

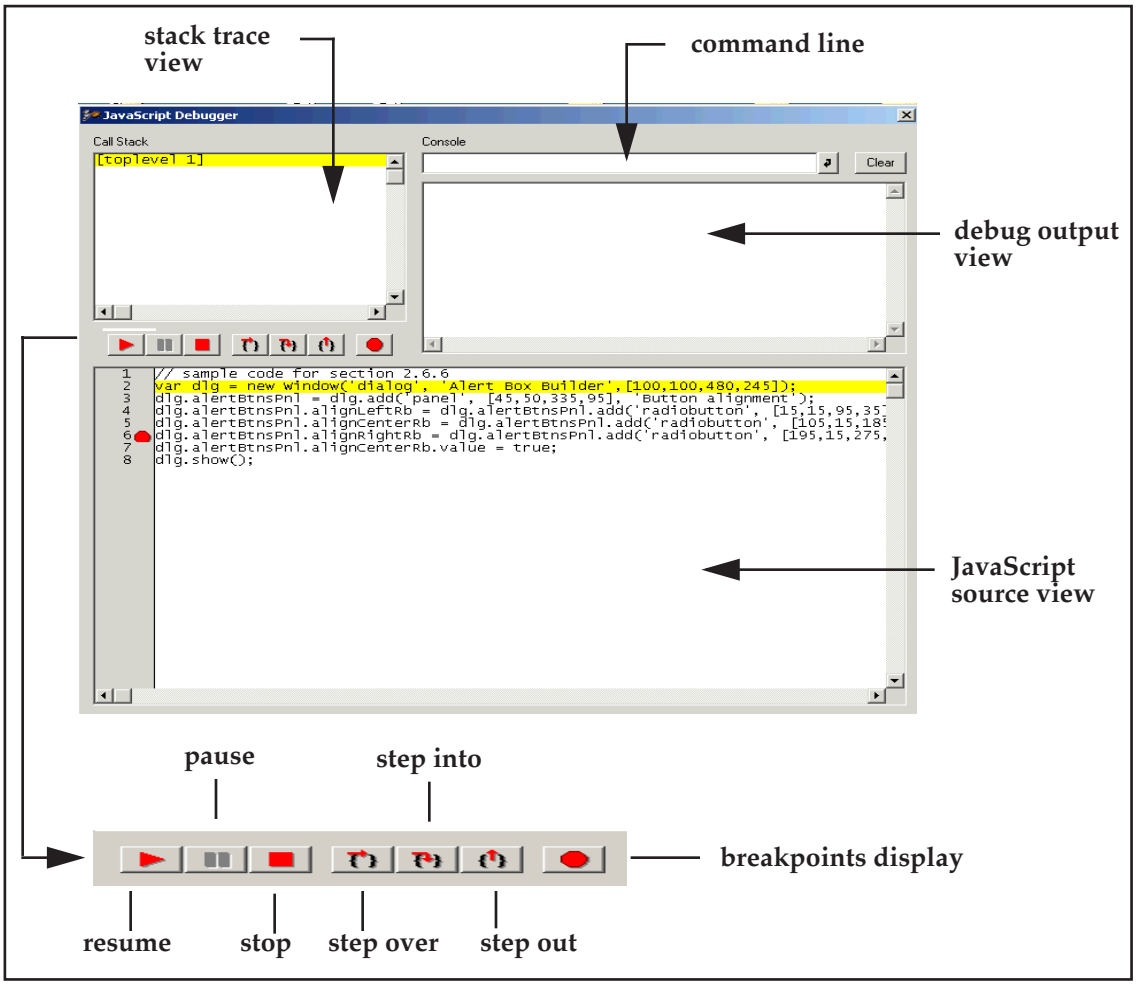
Unix encodings

On Unix systems, the File object looks for the presence of the *iconv* library, and it uses whatever encoding it finds there. If you, therefore, need a special encoding on a Unix system, make sure that there is an *iconv* encoding module installed that converts between UTF-16 (the internal format that the File object uses) and the desired encoding.

4

JavaScript Debugging

This section describes the information and controls of the JavaScript Debugger window, as illustrated below.



The Debugger Window

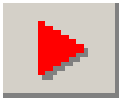
The current stack trace appears in the upper-left pane of the JavaScript Debugger window. This stack trace view displays the calling hierarchy at the time of the breakpoint. Double-clicking a line in this view changes the current scope, enabling you to inspect and modify scope specific data.

All debugging output appears in the upper-right pane of the JavaScript Debugger window. Specifically, output from the print method of the "$" object appears in this debug output view.

The currently executing JavaScript source appears in the lower pane of the JavaScript Debugger window. Double-clicking a line in this JavaScript source view sets or clears an unconditional breakpoint on that line. That is, if a breakpoint is in effect for that line, double-clicking it clears the breakpoint, and vice-versa. The line number column to the left of the source view displays a red dot for all lines with a breakpoint.

Controlling Code Execution in the JavaScript Debugger Window

This section describes the buttons that control the execution of code when the JavaScript Debugger window is active. Most of these buttons also provide a keyboard shortcut available as a *Ctrl-key* combination on Windows platforms or a *Cmd-key* combination on Mac OS platforms.



Resume

Cmd-R (Mac OS)

Ctrl-R (Windows)

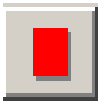
Resumes execution of the script with the JavaScript Debugger window open. When the script terminates, the application closes the JavaScript Debugger window automatically. Closing the debugger window manually also causes script execution to resume. This button is enabled when script execution is paused or stopped.

Pause

Cmd-P (Mac OS)

Ctrl-P (Windows)

Halts the currently executing script temporarily and reactivate the JavaScript Debugger window. This button is enabled when a script is running.

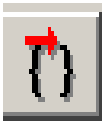


Stop

Cmd-K (Mac OS)

Ctrl-K (Windows)

Stops execution of the script and generate a runtime error. This button is enabled when a script is running.

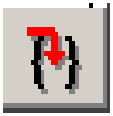


Step Over

Cmd-S (Mac OS)

Ctrl-S (Windows)

Halts after executing a single JavaScript statement in the script; if the statement calls a JavaScript function, execute the function in its entirety before stopping.

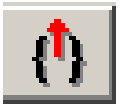


Step Into

Cmd-T (Mac OS)

Ctrl-T (Windows)

Halts after executing a single JavaScript statement in the script or after executing a single statement in any JavaScript function that the script calls.

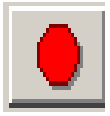


Step Out

Cmd-U (Mac OS)

Ctrl-U (Windows)

When the debugger is paused within the body of a JavaScript function, clicking this button resumes script execution until the function returns. When paused outside the body of a function, clicking this button resumes script execution until the script terminates.



Script Breakpoints Display

(no keyboard shortcuts)

Clicking this button displays the Script Breakpoints Window. Type in a line number that corresponds to a desired breakpoint and a condition about whether to stop or not. For more information on breakpoints, see “JavaScript Breakpoints Window” on page 75.

Using the JavaScript Command Line Entry Field

You can use the JavaScript Debugger window's command line entry field to enter and execute JavaScript code interactively within a specified stack scope. Commands entered in this field execute with a time-out of one second. If a command takes longer than one second to execute, the script terminates and generates a time-out error.

Command line entry field.

Enter in this field a JavaScript statement to execute within the stack scope of the line highlighted in the Stack Trace view. When you've finished entering the JavaScript expression, you can execute it by clicking the command line entry button or pressing the Enter key. Click the button next to the field or press Enter to execute the JavaScript code in the command line entry field. The application executes the contents of the command line entry field within the stack scope of the line highlighted in the Stack Trace view.

The command line entry field accepts any JavaScript code, making it very convenient to use for inspecting or changing the contents of variables.

Note: To list the contents of an object as if it were JavaScript source code, enter the `object.toSource()` command, replacing object with the object that you want to display.

Setting Breakpoints

You can set breakpoints in the debugger itself, by calling methods of the $ object, or by defining them in your JavaScript code.

Setting Breakpoints in the JavaScript Debugger Window

When the JavaScript Debugger window is active, you can double-click a line in the source view to set or clear a breakpoint at that line. Alternatively, you can click the button to display the Script Breakpoints window and set or clear breakpoints in this window as described in "Setting Breakpoints in the JavaScript Breakpoints Window".

Setting Breakpoints in JavaScript Code

Adding the debugger statement to a script sets an unconditional breakpoint. For example, the following code causes the script to halt and display the script debug window as soon as it enters the setupBox function. Note that debugging must be enabled; if not, the debugger statement is ignored.

```
function setupBox(box) {
     // break unconditionally at the next line
     debugger;
     box.width  = 48;
     box.height = 48;
     box.url    = "none";
}
```

To execute a breakpoint in runtime code, call the `$.bp()` method, as shown in the following example:

```
function setupBox(box) {
     box.width  = (box.width == undefined) ? $.bp() : 48;
     box.height = (box.height == undefined) ? $.bp() : 48;
     box.url    = (box.url == undefined) ? $.bp() : "none";
}
```

This example breaks if any of the width, height, or url attributes of the custom element are undefined. Of course, you wouldn't put bp method calls into commercial code—it's more appropriate for shipping code to set default values for undefined properties, as the previous example does. Again, debugging must be enabled; if not, the system ignores the `$.bp()` method.

You can also use `$.bp()` with a conditional JavaScript statement:

```
function setupBox(box) {
     box.width  = 48; $.bp (box.width == undefined);
     box.height = 48; $.bp (box.height == undefined);
     box.url    = "none"; $.bp (box.url == undefined);
}
```

Embedding Debugger Instructions into a Script

The JavaScript engine implements a global object, '$', which has a `.level` property useful when embedding instructions on debugger behavior directly into a script. The following values are supported:

- 0 -- Display of the debugger is suppressed (default)
- 1 -- Break on run-time errors, or when a 'debugger' statement is encountered
- 2 -- Display the debugger at the beginning of script execution

For example:

```
$.level = 1; // Show the debugger if run-time errors are thrown
```

Provided that `$.level` is set to a value greater than zero, the debugger will become activated and stop on a line containing this statement. This property effectively functions as a breakpoint statement. See "The Debugger Object ($)" on page 77 for more information.

Script Prompt

When a run-time error is encountered while debugging a script, the following dialog is typically displayed.

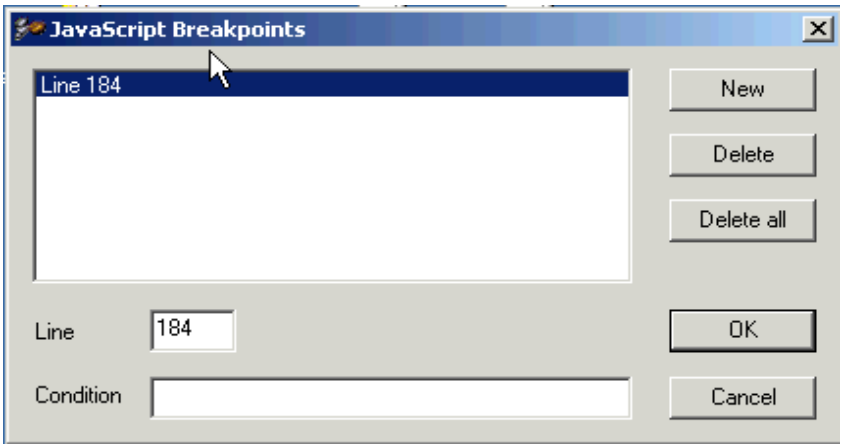
The following conditions apply:

- If *Yes* is selected, the error is ignored and script processing attempts to continue on the next executable line
- If *No* is selected, the error is handled as it would be during normal (non-debug) execution
- If the error was encountered in a script-defined *try block*, execution jumps to the first line of the try's associated catch block
- If the error was **not** encountered in a script-defined *try block*, script execution is terminated immediately

Note: To debug through the normal flow of execution in such *try block* circumstances, users typically select *No*. However, if the error is determined not to be of a nature that would compromise continued in-line execution (such as during script development), you are given the option of not having to terminate, fix or restart.

JavaScript Breakpoints Window

This section describes the information and controls that the JavaScript Breakpoints window provides. Display of the Script Breakpoints window is controlled by the JavaScript Breakpoints button in the main JavaScript Debugger Window.



This dialog displays all defined breakpoints. This dialog does not display:

- Breakpoints defined by the debugger statement in JavaScript code.
- Temporary breakpoints.

The JavaScript Breakpoints window provides the following controls:

- The Line field contains the line number of the breakpoint.
- The Condition field may contain a JavaScript expression to evaluate when the breakpoint is reached. If the expression evaluates to false, the breakpoint is not executed.

Breakpoints set in this window persist across multiple executions of a script. When the application quits, or a script is reloaded, it removes all breakpoints.

Setting Breakpoints in the JavaScript Breakpoints Window

Take the following steps to set a breakpoint in the JavaScript Breakpoints Window:

- Click the breakpoint that you wish to edit if applicable.
- Enter a line number in the Line Number field, or change the existing line number.
- Optionally, enter a condition such as "j == 1000" in the Condition field. This can be any valid JavaScript expression. If the result of evaluating the expression is true, the breakpoint

activates. The breakpoint also activates if there is a syntax or runtime error during the execution of the condition.

- Click “New” to change the line number of the breakpoint, to add or remove a breakpoint condition, or to create a new breakpoint.

The Debugger Object ($)

The $ Object (Debugger Object) provides properties and methods you can use to debug your JavaScript code. For example, you can call Debugger methods to set or clear breakpoints programmatically, or to change the language flavor of the script currently executing. It also provides properties that hold information about the version of the host platform's operating system.

Note: The $ object is not a standard JavaScript object.

Properties

Property	Type	Description
`error`	*Error*	Retrieves the last runtime error. Reading this property returns an Error object containing information about the last runtime error.
`level`	*Number*	Sets the debugging level. This may be one of three values: 0 – disable debugging. 1 – break on runtime errors. 2 – break at the beginning of the script. Note that the debugger statement is disabled as well if the debugging level is 0. Also, your scripting environment needs to support level 2 explicitly. If level 2 does not work, use the statement "$.level = 1; debugger;" at the beginning of your script.
`version`	*String*	Returns the version number of the engine as a three-part number like "3.1.11". Read only.
`os`	*String*	Outputs the current operating system version. Read only.

Debug output

```
write (text, …);
writeln (text, …);
```

Writes the given string to the Debug Output window. The writeln method appends a New Line character to its arguments.

Parameter	Type	Description	Returns
`text`	*String*	All parameters are concatenated to a single string.	Undefined

Breakpoints

```
setbp (scriptletName, line, [condition]);
```

Sets a breakpoint. The breakpoint is defined by the name of the scriptlet or function and the line number. If the name is the empty string or missing, the name of the currently executing scriptlet is used. If the line number is zero or not supplied, the current line number is used. Thus, the call `$.setbp()` without parameters sets a breakpoint at the current position.

Optionally, a condition may be supplied. This is a JavaScript expression string that is evaluated before the breakpoint is executed. The breakpoint is only executed if the expression returns true.

The special string "NEXTCALL", as the scriptlet name suggests, causes the engine to execute a breakpoint the next time a function call is executed.

Parameter	Type	Description	Returns
`scriptletName`	*String*	The name of the scriptlet or function where the breakpoint is to be set.	Undefined
`line`	*Number*	The line number where the breakpoint is to be set.	
`condition`	*String*	An optional JavaScript expression that is evaluated before the breakpoint is executed. The expression needs to evaluate to the equivalent of true in order to activate the breakpoint.	

```
clearbp (scriptletName, line);
```

Clears a breakpoint. The breakpoint is defined by the name of the scriptlet or function and the line number. If the scriptlet name is the empty string or missing, the name of the currently executing scriptlet is used. If the line number is zero or not supplied, the current line number is used. Thus, the call `$.clearbp()` without parameters clears a breakpoint at the current position.

The special string "NEXTCALL," as the scriptlet name suggests, causes the engine to clear a breakpoint at the next function call.

Parameter	Type	Description	Returns
`scriptletName`	*String*	The name of the scriptlet or function where the breakpoint is to be cleared.	Undefined
`line`	*Number*	The line number where the breakpoint is to be cleared.	

`bp([condition]);`

Executes a breakpoint at the current position. Optionally, a condition may be supplied. This is a JavaScript expression string that is evaluated before the breakpoint is executed. The breakpoint is only executed if the expression returns true. If no condition is given, the use of the debugger statement is recommended.

Parameter	Type	Description	Returns
`condition`	*String*	An optional JavaScript expression string that is evaluated before the breakpoint is executed. The expression needs to evaluate to the equivalent of true in order to activate the breakpoint.	Undefined

Other methods

`gc()`

Initiates garbage collection. Garbage collection is a convenience function that automatically collects all varibles declared as `var`. This method allow you to **manually** invoke garbage collection.

Returns: *Undefined*

5

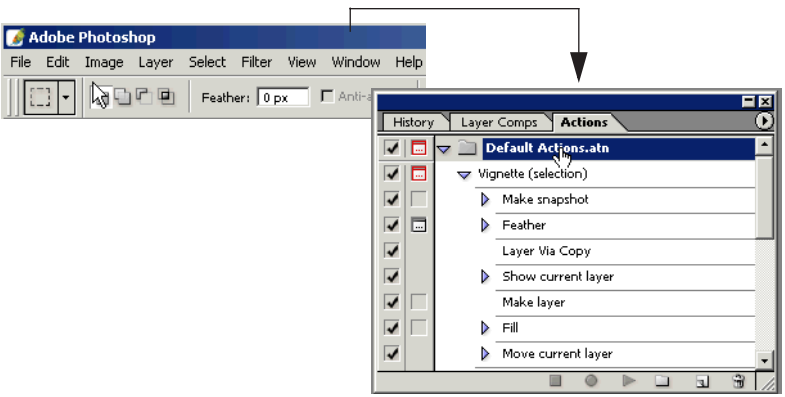
Utilities

Photoshop actions are different from scripts. A Photoshop action is a series of tasks you have recorded while using the application—menu choices, tool choices, selection, and other commands. When you "play" an action, Photoshop performs all of the recorded commands.

Actions and scripts are two ways of automating repetitive tasks, but they work very differently.

- You cannot add conditional logic to an action. Unlike a script, actions cannot make decisions based on the current situation.
- A single script can target multiple hosts. Actions can't. For example, you could target both Photoshop and Illustrator in the same script.

The Actions palette, invoked under the Window menu, supports actions with a great deal of sophistication (including the ability to display dialogs) and allows users to work with selected objects, as illustrated below.



The action manager is a Photoshop CS utility that helps you manage and manipulate actions when writing JavaScripts.

Action Manager scripting

In addition to accessing Action Manager from the *palette*, you can incorporate Action Manager functionality into your *scripts*. Moreover, the Action Manager allows you to write scripts that target Photoshop functionality that is not otherwise accessible. You are able to script third party plug-ins, filters, and other tasks that are not otherwise included in the scripting interface. The only requirement is that the task that you want to access from the Action Manager is recordable.

The classes "ActionDescriptor", "ActionReference" and "ActionList" are all part of the Action Manager functionality.

When you write scripts that use the Action Manager, you should install the "ScriptingListener" plug-in. This plug-in is located in the "Scripting Guide" folder. Look inside the "utilities" folder that is part of the scripting support download.

To install the plug-in place it in the Adobe `Photoshop CS\Plug-Ins\Adobe Photoshop Only\Automate\` folder.

"ScriptingListener" records most of your actions to a file on your hard drive. To avoid slowing down Photoshop as well as not to create a big file on your drive, only install the plug-in when you are creating Action Manager scripts.

When "ScriptingListener" is installed it will record a file with scripting code corresponding to the actions that you perform from the UI.

The Windows version of "ScriptingListener" creates the following file:

- **C:\ScriptingListenerJS.log**: contains JavaScript code corresponding to the actions that are performed from the UI.

The Macintosh version "ScriptingListener" creates the following file:

- **ScriptingListenerJS.log**: the file is created on the desktop and contains JavaScript code corresponding to the actions that are performed from the UI.

Note: There is no AppleScript interface to the Action Manager, but you can execute JavaScripts from AppleScript, so you are able to access Action Manager functionality from AppleScripts.

Using the Action Manager from JavaScript

Imagine, for example, that you want to be able to use the Emboss filter. The Emboss filter is not part of the filters that are exposed to the various scripting languages, but using the Action Manager you are able to use this filter. First make sure that you have installed the "ScriptingListener". Then from the UI, open a document and apply the Emboss filter using the settings: angle 135, height 3 and amount 100.

When the ScriptingListener is installed, running the Emboss filter is recorded to a file called "ScriptingListenerJS.log" (see above for location of this file on the various platforms).

Open the "ScriptingListenerJS.log" file. At the end of the file you will see something like the following. Note the numbers may vary:

```
var id19 = charIDToTypeID( "Embs" );
var desc4 = new ActionDescriptor();
var id20 = charIDToTypeID( "Angl" );
desc4.putInteger( id20, 135 );
var id21 = charIDToTypeID( "Hght" );
desc4.putInteger( id21, 3 );
var id22 = charIDToTypeID( "Amnt" );
desc4.putInteger( id22, 100 );
executeAction( id19, desc4 );
```

The ScriptingListener divides every command by a line, so it is easy to find the last command.

The next step in making Emboss scriptable is to identify the values that you entered (135, 3 and 100). Copy the JavaScript code from the "ScriptingListenerJS.log" file to another file and substitute the filter values with variable names. In the following example, we have wrapped the code in a JavaScript function and replaced 135 with angle, 3 with height, and 100 with amount.

```
function emboss( angle, height, amount )
{
var id32 = charIDToTypeID( "Embs" );
var desc7 = new ActionDescriptor();
var id33 = charIDToTypeID( "Angl" );
desc7.putInteger( id33, angle );
var id34 = charIDToTypeID( "Hght" );
desc7.putInteger( id34, height );
var id35 = charIDToTypeID( "Amnt" );
desc7.putInteger( id35, amount );
executeAction( id32, desc7 );
     }
```

You now have a JavaScript function that performs the Emboss filter on the current document. To activate the Emboss filter from JavaScript you must include the function definition shown above and then call the function with the desired parameters. To apply Emboss with angle 75, height 2 and amount 89, you say:

```
// First include the emboss function somewhere in your JavaScript
// file
function emboss( angle, height, amount )
{
var id32 = charIDToTypeID( "Embs" );
var desc7 = new ActionDescriptor();
var id33 = charIDToTypeID( "Angl" );
desc7.putInteger( id33, angle );
```

```
var id34 = charIDToTypeID( "Hght" );
desc7.putInteger( id34, height );
var id35 = charIDToTypeID( "Amnt" );
desc7.putInteger( id35, amount );
executeAction( id32, desc7 );
}
// Then call emboss with desired parameters
emboss( 75, 2, 89 );
```

Running JavaScript based Action Manager code from AppleScript

As there is no Action Manager functionality in AppleScript you will have to use JavaScript to use the Action Manager on the Mac. To do this you use the AppleScript command: "do javascript." Provide filter settings in the "arguments" parameter.

You need to re-write your JavaScript code slightly to work with the "do javascript" command to use the "arguments" collection to get access to the AppleScript values from JavaScript. For example change the Emboss JavaScript shown in the previous section to the following and save it in a file called "Emboss.js":

```
function emboss( angle, height, amount )
{
var id32 = charIDToTypeID( "Embs" );
var desc7 = new ActionDescriptor();
var id33 = charIDToTypeID( "Angl" );
desc7.putInteger( id33, angle );
var id34 = charIDToTypeID( "Hght" );
desc7.putInteger( id34, height );
var id35 = charIDToTypeID( "Amnt" );
desc7.putInteger( id35, amount );
executeAction( id32, desc7 );
}
// Call emboss with values provided in the "arguments" collection
emboss( arguments[0], arguments[1], arguments[2] );
From AppleScript you can then run the Emboss filter by saying:
tell application "Adobe Photoshop CS"
do javascript (file <a path to Emboss.js>) ¬
    with arguments { 75,2,89 }
end tell
```

Running JavaScript based Action Manager code from VBScript

From VBScript you have a choice of either running JavaScript based Action Manager code or VBScript based Action Manager code. This section describes how to access JavaScript based Action Manager code. The next section covers how to run VBScript based Action Manager code.

To access JavaScript code from VBScript, you must use the “DoJavaScriptFile” command and provide specific settings in the “arguments” parameter.

Save the following script in a file called “C:\Emboss.js”

```
function emboss( angle, height, amount )
{
var id32 = charIDToTypeID( "Embs" );
var desc7 = new ActionDescriptor();
var id33 = charIDToTypeID( "Angl" );
desc7.putInteger( id33, angle );
var id34 = charIDToTypeID( "Hght" );
desc7.putInteger( id34, height );
var id35 = charIDToTypeID( "Amnt" );
desc7.putInteger( id35, amount );
executeAction( id32, desc7 );
}
// Call emboss with values provided in the "arguments" collection
emboss( arguments[0], arguments[1], arguments[2] );
```

From VBScript you can then run the Emboss filter by saying:

```
Set objApp = CreateObject("Photoshop.Application")
objApp.DoJavaScriptFile "C:\Emboss.js", Array(75, 2, 89)
```

6

JavaScript Interface

The object classes of the JavaScript type library are presented alphabetically and in tabular format in this interface section.

Class *properties* and *methods* are described.

- Properties of a class include the property itself, access status (read only or read/write) , value type, and a description.

  When the value type is an enumeration, enumerated values are defined in UPPER CASE, as illustrated below:

Property	Access	Value Type	What it is
displayDialogs	R/W	DialogModes DialogModes.ALL DialogModes.ERROR DialogModes.NO	controls whether or not Photoshop displays dialogs

- Methods of a class include the method name, a description, parameters and return values, if applicable.

  Italics indicates that a parameter type is **optional**.

  When the parameter type is an enumeration, enumerated values are defined in UPPER CASE.

Method	What it does	Parameter Type	Returns
purge	Purges one or more caches	target as PurgeTarget PurgeTarget.ALLCACHES PurgeTarget.CLIPBOARDCACHE PurgeTarget.HISTORYCACHES PurgeTarget.UNDOCACHES	

Note: Descriptions are omitted for properties and methods that are self-explanatory; for example: *removeAll*. Return values and parameter types, if none apply, may also be left **blank**.

Sample code for several object model classes is given to help illustrate the syntax as well as usage of the object class.

ActionDescriptor

Properties

Property	Access	Value Type	What it is
count	RO	Long	number of keys contained in the descriptor

Methods

Method	What it does	Parameter Type	Returns
clear	Clears the descriptor		
erase	Erases a key from the descriptor	Key as Long	
getBoolean	Gets the value of a key of type boolean	Key as Long	Boolean
getClass	Gets the value of a key of type class	Key as Long	Long
getDouble	Gets the value of a key of type double	Key as Long	Double
getEnumerationType	Gets the enumeration type of a key	Key as Long	Long
getEnumerationValue	Gets the enumeration value of a key	Key as Long	Long
getInteger	Gets the value of a key of type integer	Key as Long	Long
getKey	Gets ID of the Nth key	Index as Long	Long
getList	Gets the value of a key of type list	Key as Long	ActionList
getObjectType	Gets the class ID of an object in a key of type object	Key as Long	Long
getObjectValue	Gets the value of a key of type object	Key as Long	ActionDescriptor
getPath	Gets the value of a key of type Alias	Key as Long	File
getReference	Gets the value of a key of type ActionReference	Key as Long	ActionReference

Method	What it does	Parameter Type	Returns
getString	Gets the value of a key of type string	Key as Long	String
getType	Gets the type of a key	Key as Long	DescValueType
getUnitDoubleType	Gets the unit type of a key of type UnitDouble	Key as Long	Long
getUnitDoubleValue	Gets the value of a key of type UnitDouble	Key as Long	Double
hasKey	Does the descriptor contain the provided key?	Key as Long	Boolean
isEqual		otherDesc as ActionDescriptor	Boolean
putBoolean		Key as Long Value as Boolean	
putClass		Key as Long Value as Long	
putDouble		Key as Long Value as Double	
putEnumerated		Key as Long enumType as Long Value as Long	
putInteger		Key as Long Value as Long	
putList		Key as Long Value as ActionList	
putObject		Key as Long classID as Long Value as ActionDescriptor	
putPath		Key as Long Value as File	
putReference		Key as Long Value as ActionReference	
putString		Key as Long Value as String	
putUnitDouble		Key as Long unitID as Long Value as Double	

ActionList

Properties

Property	Access	Value Type	What it is
count	RO	Long	number of items in the list

Methods

Method	What it does	Parameter Type	Returns
clear	Clear the list		
getBoolean	Gets the value of an item of type boolean	Index as Long	Boolean
getClass	Gets the value of an item of type class	Index as Long	Long
getDouble	Gets the value of an item of type double	Index as Long	Double
getEnumerationType	Gets the enumeration type of an item	Index as Long	Long
getEnumerationValue	Gets the enumeration value of an item	Index as Long	Long
getInteger	Gets the value of an item of type integer	Index as Long	Long
getList	Gets the value of an item of type list	Index as Long	ActionList
getObjectType	Gets the class ID of an object in an item of type object	Index as Long	Long
getObjectValue	Gets the value of an item of type object	Index as Long	ActionDescriptor
getPath	Gets the value of an item of type Alias	Index as Long	File
getReference	Gets the value of an item of type ActionReference	Index as Long	ActionReference
getString	Gets the value of an item of type string	Index as Long	String
getType	Gets the type of an item	Index as Long	DescValueType

Method	What it does	Parameter Type	Returns
getUnitDoubleType	Gets the unit type of an item of type UnitDouble	Index as Long	Long
getUnitDoubleValue	Gets the value of anm item of type UnitDouble	Index as Long	Double
putBoolean		Value as Boolean	
putClass		Value as Long	
putDouble		Value as Double	
putEnumerated		enumType as Long Value as Long	
putInteger		Value as Long	
putList		Value as ActionList	
putObject		classID is Long Value as ActionDescriptor	
putPath		Value as File	
putReference		Value as ActionReference	
putString		Value as String	
putUnitDouble		classID is Long Value as Double	

ActionReference

Methods

Method	What it does	Parameter Type	Returns
getContainer			ActionReference
getDesiredClass			Long
getEnumeratedType	Gets type of enumeration of an ActionReference whose form is 'Enumerated'		Long
getEnumeratedValue	Gets value of enumeration of an ActionReference whose form is 'Enumerated'		Long
getForm	Gets form of ActionReference		ReferenceFormType
getIdentifier	Gets identifier value for an ActionReference whoxse form is 'Identifier'		Long
getIndex	Gets index value for an ActionReference whoxse form is 'Index'		Long
getName	Gets name value for an ActionReference whoxse form is 'Name'		String
getOffset	Gets offset value for an ActionReference whoxse form is 'Offset'		Long
getProperty	Gets property ID value for an ActionReference whoxse form is 'Property'		Long
putClass		desiredClass as Long	
putEnumerated		desiredClass as Long enumType as Long Value as Long	
putIdentifier		desiredClass as Long Value as Long	

Method	What it does	Parameter Type	Returns
putIndex		desiredClass as Long Value as Long	
putName		desiredClass as Long Value as String	
putOffset		desiredClass as Long Value as Long	
putProperty		desiredClass as Long Value as Long	

Application

Properties

Property	Access	Value Type	What it is
activeDocument	R/W	Document	the frontmost document
backgroundColor	R/W	SolidColor	
colorSettings	R/W	ANYTHING	name of selected color settings' set
displayDialogs	R/W	DialogModes DialogModes.ALL DialogModes.ERROR DialogModes.NO	controls whether or not Photoshop displays dialogs
documents	RO	Documents	the open documents
fonts	RO	TextFonts	the fonts installed on this system
foregroundColor	R/W	SolidColor	
freeMemory	RO	Double	the amount of unused memory available to Adobe Photoshop
name	RO	String	the application's name
path	RO	File	the full path of the location of the Photoshop application
preferences	RO	Preferences	preference settings
scriptingVersion	RO	String	the version of the Scripting interface
version	RO	String	the version of Adobe Photoshop application

Methods

Method	What it does	Parameter Type	Returns
beep			
charIDToTypeID	Converts from a four character code to a runtime ID	charID as String	Long
doAction	Plays an action from the Actions Palette	action as String from as String	

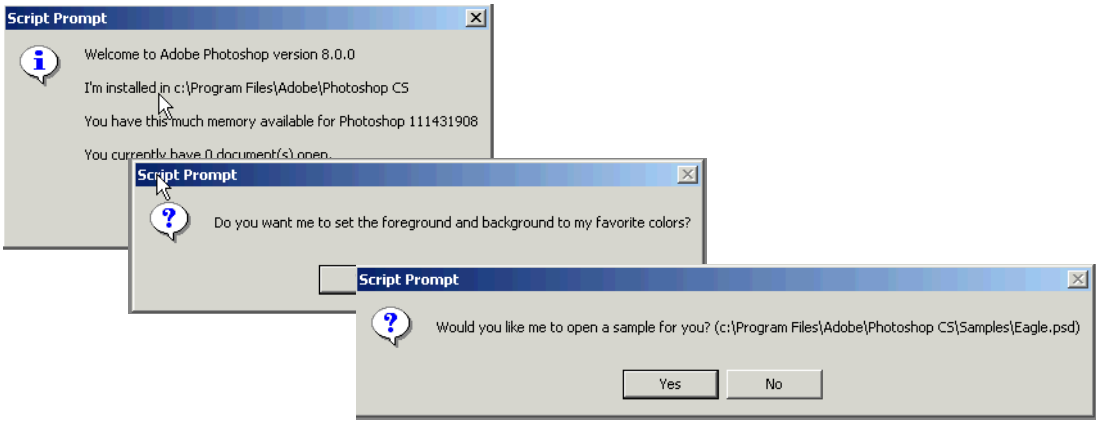
Method	What it does	Parameter Type	Returns
executeAction	Plays an ActionManager event	eventID as Long *descriptor as ActionDescriptor* *displayDialogs as DialogModes* *DialogModes.ALL* *DialogModes.ERROR* *DialogModes.NO*	ActionDescriptor
executeActionGet	Obtains an action descriptor	reference as ActionReference	ActionDescriptor
load()			
makePhotoGallery	Creates a web photo gallery	inputFolder as File outputFolder as File *options as GalleryOptions*	String
makePDFPresentation()			
open	Opens the specified document	document asFile option as *ANYTHING*	Document
purge	Purges one or more caches	target as PurgeTarget PurgeTarget.ALLCACHES PurgeTarget.CLIPBOARDCACHE PurgeTarget.HISTORYCACHES PurgeTarget.UNDOCACHES	
stringIDToTypeID	Converts from a string ID to a runtime ID	stringID as String	Long
typeIDToCharID	Converts from a runtime ID to a character ID	typeID as Long	String
typeIDToStringID	Converts from a runtime ID to a string ID	typeID as Long	String

First Sample Script

The following *application* script invokes an alert box to display properties important to an application such as version number, path to the application, memory available, and number of documents open.

Pressing the OK button on the alert box opens a second dialog, which asks users whether they would like the foreground and background colors set for the document presently open. If no document is open, the script opens a new document for the user.

The script (with no document open) produces the following progression of dialogs.



Code (application.js)

```
// build up a message to display to the user

// append the name of the application and the version
var message = "Welcome to " + app.name;
message += " version " + app.version + "\r\r";

// find out where Photoshop is installed
message += "I'm installed in " + app.path.fsName + "\r\r";

// see how much memory Photoshop has to play with
message += "You have this much memory available for Photoshop " +
app.freeMemory + "\r\r";

// see how many docments are open
var documentsOpen = app.documents.length;
message += "You currently have " + documentsOpen + " document(s) open.\r\r";

// display the message to the user
alert(message);

// answer will be true for a "Yes" answer and false for a "No" answer
var answer = confirm("Do you want me to set the foreground and background to my
favorite colors?");

// set the colors
if (answer) {
```

```
    // I don't have a favorite color. Why did I ask you may wonder?
    app.foregroundColor.rgb.red = Math.random() * 255;
    app.foregroundColor.rgb.green = Math.random() * 255;
    app.foregroundColor.rgb.blue = Math.random() * 255;

    app.backgroundColor.rgb.red = Math.random() * 255;
    app.backgroundColor.rgb.green = Math.random() * 255;
    app.backgroundColor.rgb.blue = Math.random() * 255;
}

// You really need a document open
if (app.documentsOpen == 0) {

    // use the application's path and the offset to the samples folder
    var sampleDocToOpen = File(app.path + "/Samples/Eagle.psd");

    // compose a message with the name of the file
    message = "Would you like me to open a sample for you? (";
    message += sampleDocToOpen.fsName;
    message += ")";

    // ask the user another question
    answer = confirm(message);

    // open the document accordingly
    if (answer) {
        open(sampleDocToOpen);
    }
}
```

Second Sample Script

The following *PDF presentation* script presents a slide show images in PDF format.

The script produces the following progression of images as a PDF slide show.

Code (PDFPresentation.js)

```
// use all the files in the Samples folder
var inputFolder = new Folder(app.path + "/Samples/");

// see if we got something interesting
if ( inputFolder != null) {

    // get all the files found in this folder that are Photoshop (.psd)
    var inputFiles = inputFolder.getFiles("*.psd");

    // output to the desktop
    var outputFile = File("~/Desktop/JavaScriptPresentation.pdf");

    // there are defaults but I like to set the options myself
    var options = new PresentationOptions;
    options.presentation = true;
```

```
    options.view = true;
    options.autoAdvance = true;
    options.interval = 5;
    options.loop = true;
    options.transition = TransitionType.RANDOM;

    // create the presentation
    makePDFPresentation(inputFiles, outputFile, options);
}
```

Note: To run this code on non-English platforms, substitue the following path for the `outputFile` variable:

```
    var outputFile = File("~/JavaScriptPresentation.pdf");
```

ArtLayer

Properties

Property	Access	Value Type	What it is
allLocked	R/W	Boolean	
blendMode	R/W	BlendMode BlendMode.COLORBLEND BlendMode.COLORBURN BlendMode.COLORDODGE BlendMode.DARKEN BlendMode.DIFFERENCE BlendMode.DISSOLVE BlendMode.EXCLUSION BlendMode.HARDLIGHT BlendMode.HUE BlendMode.LIGHTEN BlendMode.LINEARBURN BlendMode.LINEARDODGE BlendMode.LINEARLIGHT BlendMode.LUMINOSITY BlendMode.MULTIPLY BlendMode.NORMAL BlendMode.OVERLAY BlendMode.PASSTHROUGH BlendMode.PINLIGHT BlendMode.SATURATION BlendMode.SCREEN BlendMode.SOFTLIGHT BlendMode.VIVIDLIGHT	
bounds	RO	Array(UnitValue)	Bounding rectangle of the Layer
fillOpacity	R/W	Double	the interior opacity of the layer (between 0.0 and 100.0)
grouped	R/W	Boolean	is the layer grouped with the layer below? Photoshop CS changed the menu name to Create/ Release Clipping Mask
isBackgroundLayer	R/W	Boolean	is the layer a background layer?

Property	Access	Value Type	What it is
kind	R/W	LayerKind LayerKind.BRIGHTNESSCONTRAST LayerKind.CHANNELMIXER LayerKind.COLORBALANCE LayerKind.CURVES LayerKind.GRADIENTFILL LayerKind.GRADIENTMAP LayerKind.HUESATURATION LayerKind.INVERSION LayerKind.LEVELS LayerKind.NORMAL LayerKind.PATTERNFILL LayerKind.POSTERIZE LayerKind.SELECTIVECOLOR LayerKind.SOLIDFILL LayerKind.TEXT LayerKind.THRESHOLD	to create a text layer set this property to 'text layer' on an empty art layer of type 'normal'
linkedLayers	RO	Object	
name	R/W	String	the name of the layer
opacity	R/W	Double	master opacity of layer (0.0 - 100.0)
parent	RO	Object	the object's container
pixelsLocked	R/W	Boolean	
positionLocked	R/W	Boolean	
textItem	RO	TextItem	the text item that is associated with the art layer. Only valid for art layers whose 'has text' is true
transparentPixelsLocked	R/W	Boolean	
visible	R/W	Boolean	

Methods

Method	What it does	Parameter Type	Returns
adjustBrightnessContrast	Adjusts brightness and contrast	brightness as Long contrast as Long	
adjustColorBalance		*shadows as Object* *midtones as Object* *highlights as Object* *preserveLuminosity as Boolean*	
adjustCurves	Adjusts curves of the selected channels	curveShape as Object	
adjustLevels	Adjusts levels of the selected channels	inputRangeStart as Long inputRangeEnd as Long inputRangeGamma as Double outputRangeStart as Long outputRangeEnd as Long	
applyAddNoise	Applies the add noise filter	amount as Double distribution as NoiseDistribution NoiseDistribution.GAUSSIAN NoiseDistribution.UNIFORM monochromatic as Boolean	
applyBlur	Applies the blur filter		
applyBlurMore	Applies the blur more filter		
applyClouds	Applies the clouds filter		
applyCustomFilter	Applies the custom filter	characteristics as Object scale as Long offset as Long	
applyDeInterlace	Applies the De-Interlace filter	eliminateFields as EliminateFields EliminateFields.EVENFIELDS EliminateFields.ODDFIELDS createFields as CreateFields CreateFields.DUPLICATION CreateFields.INTERPOLATION	
applyDespeckle	Applies the despeckle filter		

Method	What it does	Parameter Type	Returns
applyDifferenceClouds	Applies the difference clouds filter		
applyDiffuseGlow	Applies the diffuse glow filter	graininess as Long glowAmount as Long clearAmount as Long	
applyDisplace	Applies the displace filter	horizontalScale as Long verticalScale as Long displacement as Type as DisplacementMapType DisplacementMapType.STRETCHTOFIT DisplacementMapType.TILE undefinedAreass as UndefinedAreas UndefinedAreas.REPEATEDGEPIXELS UndefinedAreas.WRAPAROUND displacementMapFile as File	
applyDustAndScratches	Applies the dust and scratches filter	radius as Long threshold as Long	
applyGaussianBlur	Applies the Gaussian blur filter	radius as Double	
applyGlassEffect	Applies the glass filter	distortion as Long smoothness as Long scaling as Long *invert as Boolean* *texture as TextureType* *TextureType.BLOCKS* *TextureType.CANVAS* *TextureType.FILE* *TextureType.FROSTED* *TextureType.TINYLENS* *textureFile as File*	
applyHighPass	Applies the high pass filter	radius as Double	

Method	What it does	Parameter Type	Returns
applyLensFlare	Applies the lens flare filter	brightness as Long flareCenter as Array(UnitValue) lensType as LensType LensType.MOVIEPRIME LensType.PRIME105 LensType.PRIME35 LensType.ZOOMLENS	
applyMaximum	Applies the maximum filter	radius as Double	
applyMedianNoise	Applies the median noise filter	radius as Double	
applyMinimum	Applies the minimum filter	radius as Double	
applyMotionBlur	Applies the motion blur filter	angle as Long radius as Double	
applyNTSC	Applies the NTSC colors filter		
applyOceanRipple	Applies the ocean ripple filter	size as Long magnitude as Long	
applyOffset	Applies the offset filter	horizontal as UnitValue vertical as UnitValue undefinedAreas as OffsetUndefinedAreas OffsetUndefinedAreas.REPEATEDGEPIXELS OffsetUndefinedAreas.SETTOBACKGROUND OffsetUndefinedAreas.WRAPAROUND	
applyPinch	Applies the pinch filter	amount as Long	
applyPolarCoordinates	Applies the polar coordinates filter	conversion as PolarConversionType PolarConversionType.POLARTORECTANGULAR PolarConversionType.RECTANGULARTOPOLAR	

Method	What it does	Parameter Type	Returns
applyRadialBlur	Applies the radial blur filter	amount as Long blurMethod as RadialBlurMethod RadialBlurMethod.SPIN RadialBlurMethod.ZOOM blurQuality as RadialBlurQuality RadialBlurQuality.BEST RadialBlurQuality.DRAFT RadialBlurQuality.GOOD	
applyRipple	Applies the ripple filter	amount as Long size as RippleSize RippleSize.LARGE RippleSize.MEDIUM RippleSize.SMALL	
applySharpen	Applies the sharpen filter		
applySharpenEdges	Applies the sharpen edges filter		
applySharpenMore	Applies the sharpen more filter		
applyShear	Applies the shear filter	curve as Object undefinedAreas as UndefinedAreas UndefinedAreas.REPEATEDGEPIXELS UndefinedAreas.WRAPAROUND	
applySmartBlur	Applies the smart blur filter	radius as Double threshold as Double blurQuality as SmartBlurQuality SmartBlurQuality.HIGH SmartBlurQuality.LOW SmartBlurQuality.MEDIUM mode as SmartBlurMode SmartBlurMode.EDGEONLY SmartBlurMode.NORMAL SmartBlurMode.OVERLAYEDGE	

Method	What it does	Parameter Type	Returns
applySpherize	Applies the spherize filter	amount as Long mode as SpherizeMode SpherizeMode.HORIZONTAL SpherizeMode.NORMAL SpherizeMode.VERTICAL	
applyStyle		styleName as String	
applyTextureFill	Applies the texture fill filter	textureFile as File	
applyTwirl	Applies the twirl filter	angle as Long	
applyUnSharpMask	Applies the unsharp mask filter	amount as Double radius as Double threshold as Long	
applyWave	Applies the wave filter	generatorNumber as Long minimumWavelength as Long maximumWavelength as Long minimumAmplitude as Long maximumAmplitude as Long horizontalScale as Long verticalScale as Long waveType as WaveType WaveType.SINE WaveType.SQUARE WaveType.TRIANGULAR undefinedAreas as UndefinedAreas UndefinedAreas.REPEATEDGEPIXELS UndefinedAreas.WRAPAROUND randomSeed as Long	
applyZigZag	Applies the zigzag filter	amount as Long ridges as Long style as ZigZagType ZigZagType.AROUNDCENTER ZigZagType.OUTFROMCENTER ZigZagType.PONDRIPPLES	
autoContrast	Adjusts contrast of the selected channels automatically		

Method	What it does	Parameter Type	Returns
autoLevels	Adjusts levels of the selected channels using auto levels option		
clear			
copy		*merge as Boolean*	
cut			
desaturate			
duplicate	Creates a duplicate of the object	*relativeObject as Object* *insertionLocation as ElementPlacement* *ElementPlacement.INSIDE* *ElementPlacement.PLACEATBEGINNING* *ElementPlacement.PLACEATEND* *ElementPlacement.PLACEBEFORE* *ElementPlacement.PLACEAFTER*	Object (Layer)
equalize	Equalizes the levels		
invert	Inverts the currently selected layer or channels		
link	Links the layer with another layer	with as Object (Layer)	
merge	Merges the layer down. This will remove the layer from the document. The method returns a reference to the art layer that this layer is merged into		ArtLayer
mixChannels	only valid for RGB or CMYK documents	outputChannels as Object *monochrome as Boolean*	

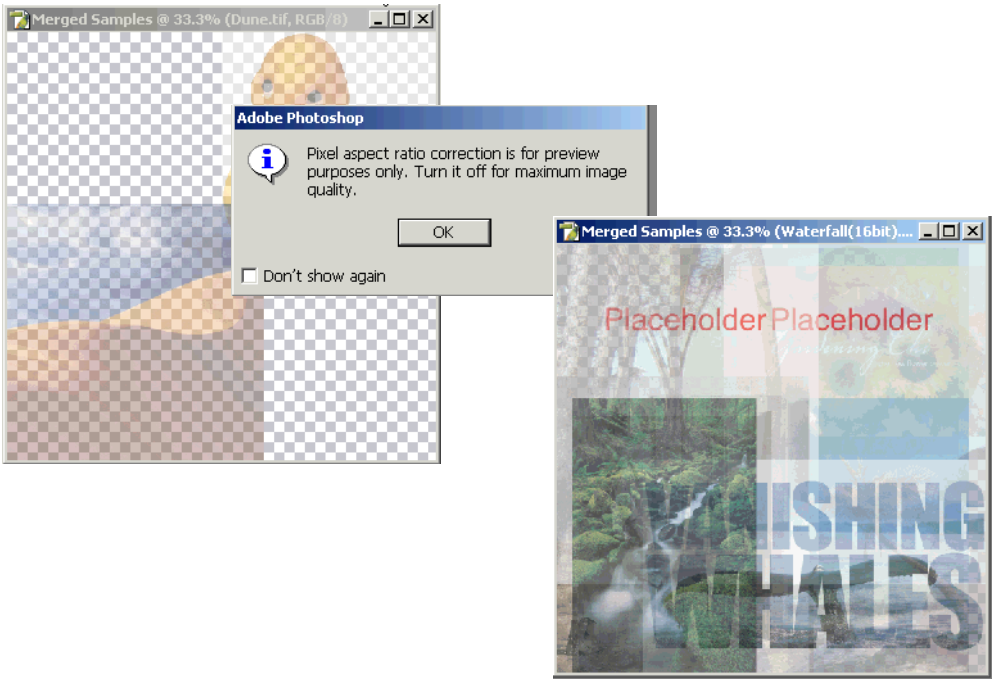
Method	What it does	Parameter Type	Returns
move	Moves the object	relativeObject as Object insertionLocation as ElementPlacement ElementPlacement.INSIDE ElementPlacement.PLACEATBEGINNING ElementPlacement.PLACEATEND ElementPlacement.PLACEBEFORE ElementPlacement.PLACEAFTER	
posterize		levels as Long	
rasterize		target as RasterizeType RasterizeType.ENTIRELAYER RasterizeType.FILLCONTENT RasterizeType.LAYERCLIPPINGPATH RasterizeType.LINKEDLAYERS RasterizeType.SHAPE RasterizeType.TEXTCONTENTS	
remove	Deletes the object		
resize		*horizontal as Double* *vertical as Double* *anchor as AnchorPosition* *AnchorPosition.BOTTOMCENTER* *.AnchorPosition.BOTTOMLEFT* *AnchorPosition.BOTTOMRIGHT* *AnchorPosition.MIDDLECENTER* *AnchorPosition.MIDDLELEFT* *AnchorPosition.MIDDLERIGHT* *AnchorPosition.TOPCENTER* *AnchorPosition.TOPLEFT* *AnchorPosition.TOPRIGHT*	
rotate		angle as Double] *anchor as AnchorPosition* *AnchorPosition.BOTTOMCENTER* *.AnchorPosition.BOTTOMLEFT* *AnchorPosition.BOTTOMRIGHT* *AnchorPosition.MIDDLECENTER* *AnchorPosition.MIDDLELEFT* *AnchorPosition.MIDDLERIGHT* *AnchorPosition.TOPCENTER* *AnchorPosition.TOPLEFT* *AnchorPosition.TOPRIGHT*	

Method	What it does	Parameter Type	Returns
selectiveColor		selectionMethod as AdjustmentReference AdjustmentReference.ABSOLUTE AdjustmentReference.RELATIVE *reds as Object* *yellows as Object* *greens as Object* *cyans as Object* *blues as Object* *magentas as Object* *whites as Object* *neutrals as Object* *blacks as Object*	
threshold		level as Long	
translate	Moves the position relative to its current position	*deltaX as UnitValue* *deltaY as UnitValue*	
unlink	Unlinks the layer		

Sample Script

The following script creates art layers to display a duck and a sand dune in an overlying checkerboard pattern. An alert box prompts the user to press OK. A multi-layered collage then displays.

The script typically produces the following progression of dialogs.



Code (ArtLayer.js)

```
// Save the current preferences
var startRulerUnits = app.preferences.rulerUnits;
var startTypeUnits = app.preferences.typeUnits;
var startDisplayDialogs = app.displayDialogs;

// Set Photoshop to use pixels and display no dialogs
app.preferences.rulerUnits = Units.PIXELS;
app.preferences.typeUnits = TypeUnits.PIXELS;
app.displayDialogs = DialogModes.NO;

// first close all the open documents
 while (app.documents.length) {
    app.activeDocument.close();
```

```
}

// create a new document to merge all the samples into
var mergedDoc = app.documents.add(1000, 1000, 72, "Merged Samples",
NewDocumentMode.RGB, DocumentFill.TRANSPARENT, 1);

// Use the path to the application and append Samples
var samplesFolder = Folder(app.path + "/Samples/");

// get all the files found in this folder
var fileList = samplesFolder.getFiles();

// open each one in turn
for (var i = 0; i < fileList.length; i++) {
    // The fileList is folders and files so open only files
    if (fileList[i] instanceof File) {
        open(fileList[i]);

        // use the document name for the layer name in the merged document
        var docName = app.activeDocument.name;

        // flatten the document so we get everything and then copy
        app.activeDocument.flatten();
        app.activeDocument.selection.selectAll();
        app.activeDocument.selection.copy();

        // don't save anything we did
        app.activeDocument.close(SaveOptions.DONOTSAVECHANGES);

        // make a random selection on the document to paste into
        // I divided the document up in 4 quadrants and I paste
        // into one of them by selecting that area
        var topLeftH = Math.floor(Math.random() * 2);
        var topLeftV = Math.floor(Math.random() * 2);
        var docH = app.activeDocument.width.value / 2;
        var docV = app.activeDocument.height.value / 2;
        var selRegion = Array(Array(topLeftH * docH, topLeftV * docV),
                          Array(topLeftH * docH + docH, topLeftV * docV),
                         Array(topLeftH * docH + docH, topLeftV * docV + docV),
                          Array(topLeftH * docH, topLeftV * docV + docV),
                          Array(topLeftH * docH, topLeftV * docV));
        app.activeDocument.selection.select(selRegion);
        app.activeDocument.paste();

        // change the layer name and muck with the opacity
        app.activeDocument.activeLayer.name = docName;
        app.activeDocument.activeLayer.fillOpacity = 50;
    }
}
```

```
// sort the layers by name
// use good old bubble sort
for (var x = 0; x < app.activeDocument.layers.length; x++) {
    for (var y = 0; y < app.activeDocument.layers.length - 1 - x; y++) {
        // Compare in a non-case sensitive way
        var doc1 = app.activeDocument.layers[y].name;
        var doc2 = app.activeDocument.layers[y + 1].name;
        if (doc1.toUpperCase() > doc2.toUpperCase()) {
            app.activeDocument.layers[y].move(app.activeDocument.layers[y+1],
                ElementPlacement.PLACEAFTER);
        }
    }
}

// Reset the application preferences
app.preferences.rulerUnits = startRulerUnits;
app.preferences.typeUnits = startTypeUnits;
app.displayDialogs = startDisplayDialogs;
```

ArtLayers

Properties

Property	Access	Value Type	What it is
length	RO	Long	number of elements in the collection
parent	RO	Object	the object's container

Methods

Method	What it does	Parameter Type	Returns
add	Creates a new object		ArtLayer
getByName	Get the first element in the collection with the provided name	name as String	ArtLayer
removeAll			

BitmapConversionOptions

Properties

Property	Access	Value Type	What it is
angle	R/W	Double	only valid for 'halftone screen' conversions
frequency	R/W	Double	only valid for 'halftone screen' conversions
method	R/W	BitmapConversionType.CUSTOMPATTERN BitmapConversionType.DIFFUIONDITHER BitmapConversionType.HALFTHRESHOLD BitmapConversionType.HALFTONESCREEN BitmapConversionType.PATTERNDITHER	(default: BitmapConversionType BitmapConversionType.--> DIFFUSIONDITHER)
patternName	R/W	only valid for 'custom pattern' conversions	String
resolution	R/W	output resolution (in pixels per inch)	Double
shape	R/W	only valid for 'halftone screen' conversions	BitmapHalfToneType

BMPSaveOptions

Properties

Property	Access	Value Type	What it is
alphaChannels	R/W	Boolean	save alpha channels
depth	R/W	BMPDepthType BMPDepthType.BMP_A1R5G5B5 BMPDepthType.BMP_A4R4G4B4 BMPDepthType.BMP_A8R8G8B8 BMPDepthType.BMP_R5G6B5 BMPDepthType.BMP_R8G8B8 BMPDepthType.BMP_X1R5G5B5 BMPDepthType.BMP_X4R4G4B4 BMPDepthType.BMP_X8R8G8B8 BMPDepthType.EIGHT BMPDepthType.FOUR BMPDepthType.ONE BMPDepthType.SIXTEEN BMPDepthType.THIRTYTWO BMPDepthType.TWENTYFOUR	number of bits per sample
flipRowOrder	R/W	Boolean	
osType	R/W	OperatingSystem OperatingSystem.OS2 OperatingSystem.WINDOWS	target OS. Windows or OS/2 (default: OperatingSystem.WINDOWS)
rleCompression	R/W	Boolean	should RLE compression be used?

Channel

Properties

Property	Access	Value Type	What it is
color	R/W	SolidColor	color of the channel (not valid for component channels)
histogram	RO	Object	color of the channel (not valid for component channels)
kind	R/W	ChannelType ChannelType.COMPONENT ChannelType.MASKEDAREA ChannelType.SELECTEDAREA ChannelType.SPOTCOLOR	type of the channel
name	R/W	String	the channel's name
opacity	R/W	Double	opacity of alpha channels (called solidity for spot channels)
parent	RO	Object	the object's container
visible	R/W	Boolean	

Methods

Method	What it does	Parameter Type	Returns
duplicate	Duplicates the channel	*targetDocument as Document*	Channel
merge	Merges a spot channel into the component channels		
remove	Deletes the object		

Channels

Properties

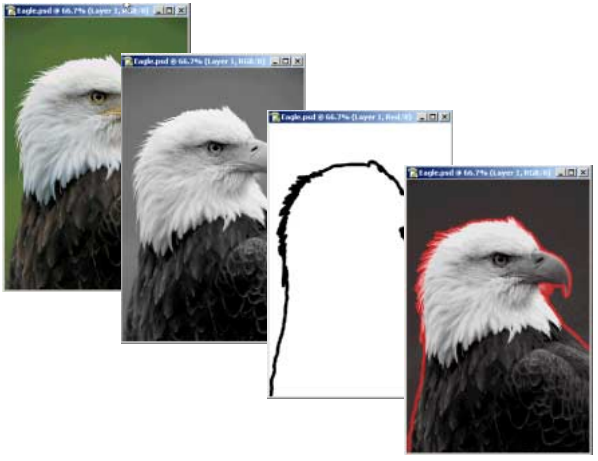
Property	Access	Value Type	What it is
length	RO	Long	number of elements in the collection
parent	RO	Object	the object's container

Methods

Method	What it does	Parameter Type	Returns
add	Creates a new object		Channel
getByName	Get the first element in the collection with the provided name	name as String	Channel
removeAll			

Sample Script

The following script produces a strobe effect, as a progression of dialogs display.



Code (Histogram.js)

```
// Save the current preferences
var startRulerUnits = app.preferences.rulerUnits;
var startTypeUnits = app.preferences.typeUnits;
var startDisplayDialogs = app.displayDialogs;

// Set Photoshop to use pixels and display no dialogs
app.preferences.rulerUnits = Units.PIXELS;
app.preferences.typeUnits = TypeUnits.PIXELS;
app.displayDialogs = DialogModes.NO;

// if there are no documents open then try to open a sample file
if (app.documents.length == 0) {
    open(File(app.path + "/Samples/Eagle.psd"));
}

// get a reference to the working document
var docRef = app.activeDocument;

// create the output file
// first figure out which kind of line feeds we need
if ($.os.search(/windows/i) != -1) {
    fileLineFeed = "windows";
} else {
    fileLineFeed = "macintosh";
}

// create the output file accordingly
fileOut = new File("~/Desktop/Histogram.log");
fileOut.lineFeed = fileLineFeed;
fileOut.open("w", "TEXT", "????");

// write out a header
fileOut.write("Histogram report for " + docRef.name);

// find out how many pixels I have
var totalCount = docRef.width.value * docRef.height.value;

// more info to the out file
fileOut.write(" with a total pixel count of " + totalCount + "\n");

// channel indexer
var channelIndex = 0;

// remember which channels are currently active
var activeChannels = app.activeDocument.activeChannels;

// document histogram only works in these modes
```

```
if (docRef.mode == DocumentMode.RGB ||
    docRef.mode == DocumentMode.INDEXEDCOLOR ||
    docRef.mode == DocumentMode.CMYK) {

   // activate the main channels so we can get the documents histogram
   TurnOnDocumentHistogramChannels(docRef);

   // Output the documents histogram
   OutputHistogram(docRef.histogram, "Luminosity", fileOut);
}

// local reference to work from
var myChannels = docRef.channels;

// loop through each channel and output the histogram
for (var channelIndex = 0; channelIndex < myChannels.length; channelIndex++) {

   // the channel has to be visible to get a histogram
   myChannels[channelIndex].visible= true;

   // turn off all the other channels
   for (var secondaryIndex = 0; secondaryIndex < myChannels.length;
          secondaryIndex++) {
      if (channelIndex != secondaryIndex) {
         myChannels[secondaryIndex].visible= false;
      }
   }

   // Use the function to dump the histogram
   OutputHistogram(myChannels[channelIndex].histogram,
      myChannels[channelIndex].name, fileOut);
}

// close down the output file
fileOut.close();

// reset the active channels
docRef.activeChannels = activeChannels;

// Reset the application preferences
app.preferences.rulerUnits = startRulerUnits;
app.preferences.typeUnits = startTypeUnits;
app.displayDialogs = startDisplayDialogs;


// Utility function that takes a histogram and name
// and dumps to the output file
function OutputHistogram(inHistogram, inHistogramName, inOutFile) {

   // find ouch which count has the largest number
```

```
// I scale everthing to this number for the output
var largestCount = 0;

// a simple indexer I can reuse
var histogramIndex = 0;

// see how many samples we have toal
var histogramCount = 0;

// search through all and find the largest single item
for (histogramIndex = 0; histogramIndex < inHistogram.length;
       histogramIndex++) {
    histogramCount += inHistogram[histogramIndex];
    if (inHistogram[histogramIndex] > largestCount)
       largestCount = inHistogram[histogramIndex];
}

// These should match
if (histogramCount != totalCount) {
    alert("Something bad is happening!");
}

// see how much each "X" is going to count as
var pixelsPerX = largestCount / 100;

// output this data to the file
inOutFile.write("One X = " + pixelsPerX + " pixels.\n");

// output the name of this histogram
inOutFile.write(inHistogramName + "\n");

// loop through all the items and output in the following format
// 001 XXXXX
// 002 XX
for (histogramIndex = 0; histogramIndex < inHistogram.length;
       histogramIndex++) {

    // I need an extra "0" for this line item to keep everything in line
    if (histogramIndex < 10)
       inOutFile.write("0");

    // I need an extra "0" for this line item to keep everything in line
    if (histogramIndex < 100)
       inOutFile.write("0");

    // output the index to file
    inOutFile.write(histogramIndex);

    // some spacing to make it look nice
    inOutFile.write(" ");
```

```
        // figure out how many X's I need
        var outputX = inHistogram[histogramIndex] / largestCount * 100;

        // output the X's
        for (var a = 0; a < outputX; a++)
            inOutFile.write("X");

        inOutFile.write("\n");
    }

    inOutFile.write("\n");
}

// Function to active all the channels according to the documents mode
// Takes a document reference for input
function TurnOnDocumentHistogramChannels(inDocument) {

    // see how many channels we need to activate
    var visibleChannelCount = 0;

    // based on the mode of the document
    switch (inDocument.mode) {

        case DocumentMode.BITMAP:
        case DocumentMode.GRAYSCALE:
        case DocumentMode.INDEXEDCOLOR:
            visibleChannelCount = 1;
            break;

        case DocumentMode.DUOTONE:
            visibleChannelCount = 2;
            break;

        case DocumentMode.RGB:
        case DocumentMode.LAB:
            visibleChannelCount = 3;
            break;

        case DocumentMode.CMYK:
            visibleChannelCount = 4;
            break;

        case DocumentMode.DUOTONE:
            visibleChannelCount = 4;
            break;

        case DocumentMode.MULTICHANNEL:
        default:
            visibleChannelCount = inDocument.channels.length + 1;
```

```
            break;
        }

        // now get the channels to activate into a local array
        var aChannelArray = new Array();

        // index for the active channels array
        var aChannelIndex = 0;

        for(var channelIndex = 0; channelIndex < inDocument.channels.length;
                channelIndex++) {
            if (channelIndex < visibleChannelCount) {
                aChannelArray[aChannelIndex++] = inDocument.channels[channelIndex];
            }
        }

        // now activate them
        inDocument.activeChannels = aChannelArray;

    }
```

CMYKColor

Properties

Property	Access	Value Type	What it is
black	R/W	Double	the black color value (between 0.0 and 100.0)
cyan	R/W	Double	the black color value (between 0.0 and 100.0)
magenta	R/W	Double	the magenta color value (between 0.0 and 100.0)
yellow	R/W	Double	the yellow color value (between 0.0 and 100.0)

DCS1_SaveOptions

Properties

Property	Access	Value Type	What it is
dCS	R/W	DCSType DCSType.COLORCOMPOSITE DCSType.GRAYSCALECOMPOSITE DCSType.NOCOMPOSITE	(default: DCSType.COLORCOMPOSITE)
embedColorProfile	R/W	Boolean	embed color profile in document
encoding	R/W	SaveEncoding SaveEncoding.ASCII SaveEncoding.BINARY SaveEncoding.JPEGHIGH SaveEncoding.JPEGLOW SaveEncoding.JPEGMAXIMUM SaveEncoding.JPEGMEDIUM	type of encoding to use for document (default: SaveEncoding.BINARY)
halftoneScreen	R/W	Boolean	include halftone screen (default: false)
interpolation	R/W	Boolean	use image interpolation (default: false)
preview	R/W	Preview Preview.EIGHTBITTIFF Preview.MACOSEIGHTBIT Preview.MACOSJPEG Preview.MACOSMONOCHROME Preview.MONOCHROMETIFF Preview.NONE	type of preview (default: Preview.MACOSEIGHTBIT)
transferFunction	R/W	Boolean	include transfer functions in document (default: false)
vectorData	R/W	Boolean	include vector data

DCS2_SaveOptions

Properties

Property	Access	Value Type	What it is
dCS	R/W	DCSType DCSType.COLORCOMPOSITE DCSType.GRAYSCALECOMPOSITE DCSType.NOCOMPOSITE	(default: DCSType.NOCOMPOSITE)
embedColorProfile	R/W	Boolean	embed color profile in document
encoding	R/W	SaveEncoding SaveEncoding.ASCII SaveEncoding.BINARY SaveEncoding.JPEGHIGH SaveEncoding.JPEGLOW SaveEncoding.JPEGMAXIMUM SaveEncoding.JPEGMEDIUM	type of encoding to use for document (default: SaveEncoding.BINARY)
halftoneScreen	R/W	Boolean	include halftone screen (default: false)
interpolation	R/W	Boolean	use image interpolation (default: false)
multiFileDCS	R/W	Boolean	(default: false)
preview	R/W	Preview Preview.EIGHTBITTIFF Preview.MACOSEIGHTBIT Preview.MACOSJPEG Preview.MACOSMONOCHROME Preview.MONOCHROMETIFF Preview.NONE	type of preview (default: Preview.MACOSEIGHTBIT)
spotColors	R/W	Boolean	save spot colors
transferFunction	R/W	Boolean	include transfer functions in document (default: false)
vectorData	R/W	Boolean	include vector data

Document

Properties

Property	Access	Value Type	What it is
activeChannels	R/W	Object	selected channels for document
activeHistoryBrushSource	R/W	HistoryState	the current history state to use with the history brush for this document
activeHistoryState	R/W	HistoryState	the current history state for this document
activeLayer	R/W	Object (Layer)	selected layer for document
artLayers	RO	ArtLayers	the top level art layers in this document
backgroundLayer	RO	ArtLayer	background layer for the document. Only valid for documents that have a background layer
bitsPerChannel	R/W	BitsPerChannelType BitsPerChannelType.EIGHT BitsPerChannelType.ONE BitsPerChannelType.SIXTEEN	number of bits per channel
channels	RO	Channels	the channels in this document
colorProfileName	R/W	String	name of color profile for document. Only valid for documents that have been assigned a color profile
colorProfileType	R/W	ColorProfile ColorProfile.CUSTOM ColorProfile.NONE ColorProfile.WORKING	type of color profile management for document
componentChannels	RO	Object	all color component channels for this document
fullName	RO	File	full path name of document
height	RO	UnitValue	height of document (unit value)
histogram	RO	Object	a histogram of values for the composite document (only for RGB, CMYK and 'Indexed colors' documents)

Property	Access	Value Type	What it is
historyStates	RO	HistoryStates	the history states associated with this document
info	RO	DocumentInfo	document information
layerComps	RO	LayerComps	the layer comps associated with this document
layers	RO	Layers	the top level layers in this document
layerSets	RO	LayerSets	the top level layer sets in this document
managed	RO	Boolean	is the document a workgroup document?
mode	RO	DocumentMode DocumentMode.BITMAP DocumentMode.CMYK DocumentMode.DUOTONE DocumentMode.GRAYSCALE DocumentMode.INDEXEDCOLOR DocumentMode.LAB DocumentMode.MULTICHANNEL DocumentMode.RGB	document mode
name	RO	String	the document's name
parent	RO	Object	the object's container
path	RO	File	the path of the document
pathItems	RO	pathItems	the art paths associated with this document
pixelAspectRatio	R/W	Double	the pixel aspect ration of the document
quickMaskMode	R/W	Boolean	is the document in the quick mask mode?
resolution	RO	Double	the resolution of the document (in pixels per inch)
saved	RO	Boolean	has the document been saved since last change?
selection	RO	Selection	the document's selection
typename	RO	String	the class name of the object
width	RO	UnitValue	width of document (unit value)
xmpMetadata	RO	xmpMetadata	

Methods

Method	What it does	Parameter Type	Returns
changeMode	Changes the mode of the document	destinationMode as ChangeMode ChangeMode.BITMAP ChangeMode.CMYK ChangeMode.GRAYSCALE ChangeMode.INDEXEDCOLOR ChangeMode.LAB ChangeMode.MULTICHANNEL ChangeMode.RGB *options as Object* *(DocumentConversionOptions)*	
close	Closes the document	*saving as SaveOptions* *SaveOptions.DONOTSAVECHANGES* *SaveOptions.PROMPTTOSAVECHANGES* *SaveOptions.SAVECHANGES*	
convertProfile	Convert sthe document from using one color profile to using an other	destinationProfile as String intent as Intent Intent.ABSOLUTECOLORIMETRIC Intent.PERCEPTUAL Intent.RELATIVECOLORIMETRIC Intent.SATURATION *blackPointCompensation as Boolean* *dither as Boolean* *Dither.DIFFUSION* *Dither.NOISE* *Dither.NONE* *Dither.PATTERN*	
crop	Crops the document	bounds as Array(UnitValue) *angle as Double* *width as UnitValue* *height as UnitValue*	
duplicate	Creates a duplicate of the object		Document
exportDocument		exportIn as File *exportAs as ExportType* *ExportType.ILLUSTRATORPATHS* *options as ExportOptionsIllustrator*	
flatten	Flattens all layers in the document		

Method	What it does	Parameter Type	Returns
flipCanvas	Flips the canvas horizontally or vertically	direction as Direction Direction.HORIZONTAL Direction.VERTICAL	
importAnnotations	Import sannotations into the document	file as File	
mergeVisibleLayers	Flattens all visible layers in the document		
paste	Pastes contents of clipboard into the document	*intoSelection as Boolean*	ArtLayer
print	Prints the document	*postScriptEncoding as PrintEncoding* *PrintEncoding.*ASCII *PrintEncoding.*BINARY *PrintEncoding.*JPEG *sourceSpace as SourceSpaceType* *SourceSpaceType.DOCUMENT* *SourceSpaceType.PROOF* *printSpace as String* *intent as Intent* *Intent.ABSOLUTECOLORIMETRIC* *Intent.PERCEPTUAL* *Intent.RELATIVECOLORIMETRIC* *Intent.SATURATION* *blackPointCompensation as Boolean*	
rasterizeAllLayers	Rasterizes all layers		
resizeCanvas	Changes the size of the canvas	*width as UnitValue* *height as UnitValue* *anchor as AnchorPosition* *AnchorPosition.BOTTOMCENTER* *.AnchorPosition.BOTTOMLEFT* *AnchorPosition.BOTTOMRIGHT* *AnchorPosition.MIDDLECENTER* *AnchorPosition.MIDDLELEFT* *AnchorPosition.MIDDLERIGHT* *AnchorPosition.TOPCENTER* *AnchorPosition.TOPLEFT* *AnchorPosition.TOPRIGHT*	

Method	What it does	Parameter Type	Returns
resizeImage	Changes the size of the image	*width as UnitValue* *height as UnitValue* *resolution as Double* *resampleMethod as ResampleMethod* *ResampleMethod.BICUBIC* *ResampleMethod.BICUBICSHARPER* *ResampleMethod.BICUBICSMOOTHER* *ResampleMethod.BILINEAR* *ResampleMethod.NEARESTNEIGHBOR* *ResampleMethod.NONE*	
revealAll	Expands document to show clipped sections		
rotateCanvas	Rotates canvas of document	angle as Double	
save	Saves the document		
saveAs	Saves the document with specific save options	saveIn as File *options as ANYTHING* *asCopy as Boolean* *extensionType as Extension* *Extension.LOWERCASE* *Extension.NONE* *Extension.UPPERCASE*	
splitChannels	Splits channels of the document		Object
trap	Applies trap to a CMYK document	width as Long	
trim		*type as TrimType* *TrimType.*BOTTOMRIGHT *TrimType.*TOPLEFT *TrimType.*TRANSPARENT *top as Boolean* *left as Boolean* *bottom as Boolean* *right as Boolean*	

Sample Script

The following script creates a document that contains two images (an eagle and a duck) obtained from the Photoshop *samples* folder.

First, a test is made to determine which image is larger. Then the smaller image is resized to match the larger image. A merged document twice as high as either image is subsequently created in order to hold the two images, one stacked on top of the other.

A selection is made on the upper part of the document to paste in the eagle. The selection process is then inverted and the duck is pasted into the lower part of the document. In the final display, the eagle is positioned over the duck.

Code (Document.js)

```
// Save the current preferences
var startRulerUnits = app.preferences.rulerUnits;
var startTypeUnits = app.preferences.typeUnits;
var startDisplayDialogs = app.displayDialogs;

// Set Photoshop to use pixels and display no dialogs
app.preferences.rulerUnits = Units.PIXELS;
app.preferences.typeUnits = TypeUnits.PIXELS;
app.displayDialogs = DialogModes.NO;

// first close all the open documents
while (app.documents.length) {
    app.activeDocument.close();
```

```
}

// Now open some samples, path is the location of the executable
var eagleDoc = open(File(app.path + "/Samples/Eagle.psd"));
var duckDoc = open(File(app.path + "/Samples/Ducky.tif"));

// Find out which document is bigger
// make the smaller document the same size
// the resize requires the document be the active/front document
if ((eagleDoc.width.value * eagleDoc.height.value) > (duckDoc.width.value *
duckDoc.height.value)) {
    app.activeDocument = duckDoc;
    duckDoc.resize(eagleDoc.width, eagleDoc.height);
} else {
    app.activeDocument = eagleDoc;
    eagleDoc.resizeImage(duckDoc.width, duckDoc.height);
}

// make a new one twice as high as two files
var mergedDoc = app.documents.add(duckDoc.width, duckDoc.height * 2,
duckDoc.resolution, "EagleOverDuck");

// copy the eagle to the top, we need to make it active first
app.activeDocument = eagleDoc;
eagleDoc.activeLayer.copy();

// paste to the merged, again making the document active
app.activeDocument = mergedDoc;

// set up a square selection for the top of the new document
var selRegion = Array(Array(0, 0),
                      Array(mergedDoc.width.value, 0),
                     Array(mergedDoc.width.value, mergedDoc.height.value / 2),
                      Array(0, mergedDoc.height.value / 2),
                      Array(0, 0));
// make the selection
mergedDoc.selection.select(selRegion);

// paste in the eagle
mergedDoc.paste();

// do the same thing for the duck
app.activeDocument = duckDoc;
duckDoc.activeLayer.copy();

app.activeDocument = mergedDoc;
mergedDoc.selection.select(selRegion);

// inverting the selection we made before gets us the bottom of the document
mergedDoc.selection.invert();
```

```
// and paste the duck
mergedDoc.paste();

// get rid of our originals without modifying them
duckDoc.close(SaveOptions.DONOTSAVECHANGES);
eagleDoc.close(SaveOptions.DONOTSAVECHANGES);

// Reset the application preferences
app.preferences.rulerUnits = startRulerUnits;
app.preferences.typeUnits = startTypeUnits;
app.displayDialogs = startDisplayDialogs;
```

DocumentInfo

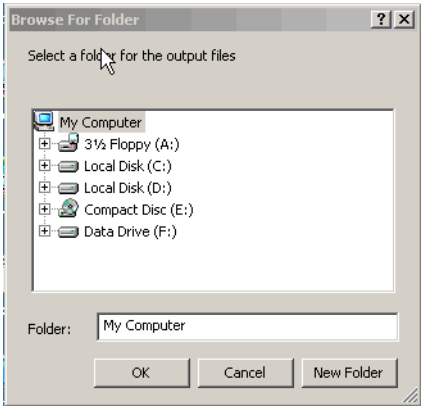
Properties

Property	Access	Value Type	What it is
author	R/W	String	
authorPosition	R/W	String	
caption	R/W	String	
captionWriter	R/W	String	
category	R/W	String	
city	R/W	String	
copyrighted	R/W	CopyrightedType CopyrightedType.COPYRIGHTEDWORK CopyrightedType.PUBLICDOMAIN CopyrightedType.UNMARKED	
copyrightNotice	R/W	String	
country	R/W	String	
creationDate	R/W	String	
credit	R/W	String	
exif	read onlly	Object	An array of 2 element arrays of type string (key/value pairs)
headline	R/W	String	
instructions	R/W	String	
jobName	R/W	String	
keywords	R/W	Object	list of keywords
ownerUrl	R/W	String	
parent	RO	Object	the object's container
provinceState	R/W	String	
source	R/W	String	
supplementalCategories	R/W	Object	
title	R/W	String	

Property	Access	Value Type	What it is
transmissionReference	R/W	String	
urgency	R/W	Urgency Urgency.FOUR Urgency.HIGH Urgency.LOW Urgency.NONE Urgency.NORMAL Urgency.SEVEN Urgency.SIX Urgency.THREE Urgency.TWO	

Sample Script

The following script opens a pop-up that allows you to select a file.



Code (DocumentInfo.js)

```
// Save the current preferences
var startDisplayDialogs = app.displayDialogs;

// Set Photoshop to use pixels and display no dialogs
app.displayDialogs = DialogModes.NO;

// ask the user for the input folder
// tag all of the documents with the photo shoot information
var inputFolder = Folder.selectDialog("Select a folder to tag");

// ask the user for the output folder
```

```
var outputFolder = Folder.selectDialog("Select a folder for the output files");

// see if we got something interesting from the dialog
if ( inputFolder != null && outputFolder != null) {

    // get all the files found in this folder
    var fileList = inputFolder.getFiles();

    // save the output's in JPEG with quality really low for small files
    var jpegOptions = new JPEGSaveOptions();

    // and I mean really small
    jpegOptions.quality = 1;

    // open each one in turn
    for (var i = 0; i < fileList.length; i++) {

        // The fileList is folders and files so open only files
        if (fileList[i] instanceof File && fileList[i].hidden == false) {

            // get a reference to our new document
            var docRef = open(fileList[i]);

            // set the file info
            docRef.info.author = "Mr. Adobe Programmer";
            docRef.info.caption = "Adobe Photo shoot";
            docRef.info.captionWriter = "Mr. Adobe Programmer";
            docRef.info.city = "San Jose";
            docRef.info.copyrightNotice = "Copyright (c) Adobe Programmer
               Photography";
            docRef.info.copyrighted = CopyrightedType.COPYRIGHTEDWORK;
            docRef.info.country = "USA";
            docRef.info.provinceState = "CA";

            // change the date to a Photoshop date format
            // "YYYYMMDD"
            var theDate = new Date();

            // the year is from 1900 ????
            var theYear = (theDate.getYear() + 1900).toString();

            // convert the month from 0..12 to 00..12
            var theMonth = theDate.getMonth().toString();

            if (theDate.getMonth() < 10) {
               theMonth = "0" + theMonth;
            }

            // convert the day from 0..31 to 00.31
            var theDay = theDate.getDate().toString();
```

```
            if (theDate.getDate() < 10) {
               theDay = "0" + theDay;
            }

            // stick them all together
            docRef.info.creationDate = theYear + theMonth + theDay;

            // flatten, we are saving to JPEG
            docRef.flatten();

            // go to 8 bit, we are saving to JPEG
            docRef.bitsPerChannel = BitsPerChannelType.EIGHT;

            // save and close
            docRef.saveAs(new File(outputFolder + "/Output" + i + ".jpg"),
               jpegOptions);

            // don't modify the original
            docRef.close(SaveOptions.DONOTSAVECHANGES);
         }
      }
   }

// Reset the application preferences
app.displayDialogs = startDisplayDialogs;
```

Documents

Properties

Property	Access	Value Type	What it is
length	RO	Long	number of elements in the collection
parent	RO	Object	the object's container

Methods

Method	What it does	Parameter Type	Returns
add	Adds a document	*width as UnitValue* *height as UnitValue* *resolution as Double* *name as String* *mode as NewDocumentMode* *NewDocumentMode.BITMAP* *NewDocumentMode.CMYK* *NewDocumentMode.GRAYSCALE* *NewDocumentMode.LAB* *NewDocumentMode.RGB* *initialFill as DocumentFill* *DocumentFill.BACKGROUNDCOLOR* *DocumentFill.TRANSPARENT* *DocumentFill.WHITE* *pixelAspectRatio as Double*	Document
getByName	Get the first element in the collection with the provided name	name as String	Document

EPSOpenOptions

Properties

Property	Access	Value Type	What it is
antiAlias	R/W	Boolean	use antialias?
constrainProportions	R/W	Boolean	constrain proportions of image
height	R/W	UnitValue	height of image (unit value)
mode	R/W	OpenDocumentMode OpenDocumentMode.CMYK OpenDocumentMode.GRAYSC OpenDocumentMode.ALE OpenDocumentMode.LAB OpenDocumentMode.RGB	the document mode
resolution	R/W	Double	the resolution of the document (in pixels per inch)
width	R/W	UnitValue	width of image (unit value)

EPSSaveOptions

Properties

Property	Access	Value Type	What it is
embedColorProfile	R/W	Boolean	embed color profile in document
encoding	R/W	SaveEncodingSaveEncoding.ASCII SaveEncoding.BINARY SaveEncoding.JPEGHIGH SaveEncoding.JPEGLOW SaveEncoding.JPEGMAXIMUM SaveEncoding.JPEGMEDIUM	type of encoding to use for document (default: SaveEncoding.BINARY)
halftoneScreen	R/W	Boolean	include halftone screen (default: false)
interpolation	R/W	Boolean	use image interpolation (default: false)
preview	R/W	Preview Preview.EIGHTBITTIFF Preview.MACOSEIGHTBIT Preview.MACOSJPEG Preview.MACOSMONOCHROME Preview.MONOCHROMETIFF Preview.NONE	type of preview
psColorManagement	R/W	Boolean	use Postscript color management (default: false)
transferFunction	R/W	Boolean	include transfer functions in document (default: false)
transparentWhites	R/W	Boolean	only valid when saving BitMap documents
vectorData	R/W	Boolean	include vector data

ExportOptionsIllustrator

Properties

Property	Access	Value Type	What it is
path	R/W	IllustratorPathType IllustratorPathType.ALLPATHS IllustratorPathType.DOCUMENTBOUNDS IllustratorPathType.NAMEDPATH	which path to export (default: IllustratorPathType.DOCUMENTBOUNDS)
pathName	R/W	String	name of path to export. Only valid if you are exporting a named path

GalleryBannerOptions

Properties

Property	Access	Value Type	What it is
contactInfo	R/W	String	web photo gallery contact info
date	R/W	String	web photo gallery date
font	R/W	GalleryFontType GalleryFontType.ARIAL GalleryFontType.COURIERNEW GalleryFontType.HELVETICA GalleryFontType.TIMESNEWROMAN	the font setting for the banner text (default: GalleryFontType.ARIAL)
fontSize	R/W	Long	the size of the font for the banner text (1 - 7; default: 3)
photographer	R/W	String	web photo gallery photographer (default:)
siteName	R/W	String	web photo gallery site name (default: Adobe Web Photo Gallery)

GalleryCustomColorOptions

Properties

Property	Access	Value Type	What it is
activeLinkColor	R/W	RGBColor	active link color
backgroundColor	R/W	RGBColor	background color
bannerColor	R/W	RGBColor	banner color
linkColor	R/W	RGBColor	link color
textColor	R/W	RGBColor	text color
visitedLinkColor	R/W	RGBColor	visited link color

GalleryImagesOptions

Properties

Property	Access	Value Type	What it is
border	R/W	Long	the amount of border pixels you want between your images (0 - 99; default: 0)
caption	R/W	Boolean	generate a caption for the images (default: false)
dimension	R/W	Long	resized image dimensions in pixels (default: 350)
font	R/W	GalleryFontType GalleryFontType.ARIAL GalleryFontType.COURIERNEW GalleryFontType.HELVETICA GalleryFontType.TIMESNEWROMAN	font for the gallery images text (default: GalleryFontType.ARIAL)
fontSize	R/W	Long	font size for the gallery images text (1 - 7; default: 3)
imageQuality	R/W	Long	the quality setting for the JPEG image (0 - 12; default: 5)
includeCopyright	R/W	Boolean	include the copyright in the text for the gallery images (default: false)
includeCredits	R/W	Boolean	include the credits in the text for the gallery images (default: false)
includeFilename	R/W	Boolean	include the file name in the text for the gallery images (default: true)
includeTitle	R/W	Boolean	include the title in the text for the gallery images (default: false)
numericLinks	R/W	Boolean	add numeric links (default: true)
resizeConstraint	R/W	GalleryConstrainType GalleryConstrainType.CONSTRAINBOTH GalleryConstrainType.CONSTRAINHEIGHT GalleryConstrainType.CONSTRAINWIDTH	how should the image be constrained (default: GalleryConstrainType.CONSTRAINBOTH)
resizeImages	R/W	Boolean	resize images data (default: true)

GalleryOptions

Properties

Property	Access	Value Type	What it is
addSizeAttributes	R/W	Boolean	add width and height attributes for images (default: true)
bannerOptions	R/W	GalleryBannerOptions	options related to banner settings
customColorOptions	R/W	GalleryCustomColorOptions	options related to custom color settings
emailAddress	R/W	String	the email address to show on the web page (default:)
imagesOptions	R/W	GalleryImagesOptions	options related to images settings
includeSubFolders	R/W	Boolean	include all files found in sub folders of the input folder (default: true)
layoutStyle	R/W	String	the style to use for laying out the web page (default: Centered Frame 1 - Basic)
preserveAllMetadata	R/W	Boolean	save all of the metadata in the JPEG files (default: false)
securityOptions	R/W	GallerySecurityOptions	options related to security settings
thumbnailOptions	R/W	GalleryThumbnailOptions	options related to thumbnail settings
useShortExtension	R/W	Boolean	short web page extension .htm or long web page extension .html (default: true)
useUTF8Encoding	R/W	Boolean	web page should use UTF-8 encoding (default: false)

GallerySecurityOptions

Properties

Property	Access	Value Type	What it is
content	R/W	GallerySecurityType GallerySecurityType.CAPTION GallerySecurityType.COPYRIGHT GallerySecurityType.CREDIT GallerySecurityType.CUSTOMTEXT GallerySecurityType.FILENAME GallerySecurityType.NONE GallerySecurityType.TITLE	web photo gallery security content (default: GallerySecurityType.NONE)
font	R/W	GalleryFontType GalleryFontType.ARIAL GalleryFontType.COURIERNEW GalleryFontType.HELVETICA GalleryFontType.TIMESNEWROMAN	web photo gallery security font (default: GalleryFontType.ARIAL)
fontSize	R/W	Long	web photo gallery security font size (1 - 72; default: 3)
opacity	R/W	Long	web page security opacity as a percent (default: 100)
text	R/W	String	web photo gallery security custom text
textColor	R/W	RGBColor	web page security text color
textPosition	R/W	GallerySecurityTextPositionType GallerySecurityTextPositionType.CENTERED GallerySecurityTextPositionType.LOWERLEFT GallerySecurityTextPositionType.LOWERRIGHT GallerySecurityTextPositionType.UPPERLEFT GallerySecurityTextPositionType.UPPERRIGHT	web photo gallery security text position (default: GallerySecurityTextPositionType.CENTERED)
textRotate	R/W	GallerySecurityTextRotateType GallerySecurityTextRotateType.CLOCKWISE45 GallerySecurityTextRotateType.CLOCKWISE90 GallerySecurityTextRotateType.COUNTERCLOCKWISE45 GallerySecurityTextRotateType.COUNTERCLOCKWISE90 GallerySecurityTextRotateType.ZERO	web photo gallery security text rotate (default: GallerySecurityTextRotateType.ZERO)

GalleryThumbnailOptions

Properties

Property	Access	Value Type	What it is
border	R/W	Long	the amount of border pixels you want around your thumbnail images (0 - 99; default: 0)
caption	R/W	Boolean	with caption (default: false)
columnCount	R/W	Long	web photo gallery thumbnail columns (default: 5)
dimension	R/W	Long	web photo gallery thumbnail dimension in pixels (default: 75)
font	R/W	GalleryFontType GalleryFontType.ARIAL GalleryFontType.COURIERNEW GalleryFontType.HELVETICA GalleryFontType.TIMESNEWROMAN	web photo gallery font (default: GalleryFontType.ARIAL)
fontSize	R/W	Long	the size of the font for the thumbnail images text (1 - 7; default: 3)
includeCopyright	R/W	Boolean	include copyright for thumbnail (default: false)
includeCredits	R/W	Boolean	include credits for thumbnail (default: false)
includeFilename	R/W	Boolean	include file name for thumbnail (default: false)
includeTitle	R/W	Boolean	include title for thumbnail (default: false)
rowCount	R/W	Long	web photo gallery thumbnail rows (default: 3)
size	R/W	GalleryThumbSizeType GalleryThumbSizeType.CUSTOM GalleryThumbSizeType.LARGE GalleryThumbSizeType.MEDIUM GalleryThumbSizeType.SMALL	the size of the thumbnail images (default: GalleryThumbSizeType.MEDIUM)

GIFSaveOptions

Properties

Property	Access	Value Type	What it is
colors	R/W	Long	number of colors in palette (only settable for some palette types)
dither	R/W	Dither Dither.DIFFUSION Dither.NOISE Dither.NONE Dither.PATTERN	type of dither
ditherAmount	R/W	Long	amount of dither. Only valid for diffusion (1 - 100; default: 75)
forced	R/W	ForcedColors ForcedColors.BLACKWHITE ForcedColors.NONE ForcedColors.PRIMARIES ForcedColors.WEB	
interlaced	R/W	Boolean	should rows be interlaced? (default: false)
matte	R/W	MatteType MatteType.BACKGROUND MatteType.BLACK MatteType.FOREGROUND MatteType.NETSCAPENONE MatteType.SEMIGRAY MatteType.WHITE	
palette	R/W	Palette Palette.EXACT Palette.LOCALADAPTIVE Palette.LOCALPERCEPTUAL Palette.LOCALSELECTIVE Palette.MACOSPALETTE Palette.MASTERADAPTIVE Palette.MASTERPERCEPTUAL Palette.MASTERSELECTIVE Palette.PREVIOUSPALETTE Palette.UNIFORM Palette.WEBPALETTE Palette.WINDOWSPALETTE	(default: Palette.LOCALSELECTIVE)
preserveExactColors	R/W	Boolean	
transparency		Boolean	

GrayColor

Properties

Property	Access	Value Type	What it is
gray	R/W	Double	the gray value (0.0 - 100.0; default: 0.0)

HistoryState

Properties

Property	Access	Value Type	What it is
name	RO	String	the channel's name
parent	RO	Object	the object's container
snapshot	RO	Boolean	is the history state a snapshot?

HistoryStates

Properties

Property	Access	Value Type	What it is
length	RO	Long	number of elements in the collection
parent	RO	Object	the object's container

Methods

Method	What it does	Parameter Type	Returns
getByName	Get the first element in the collection with the provided name	name as String	HistoryState

HSBColor

Properties

Property	Access	Value Type	What it is
brightness	R/W	Double	the brightness value (between 0.0 and 100.0)
hue	R/W	Double	the hue value (between 0.0 and 360.0)
saturation	R/W	Double	the saturation value (between 0.0 and 100.0)

IndexedConversionOptions

Properties

Property	Access	Value Type	What it is
colors	R/W	Long	number of colors in palette (only settable for some palette types)
dither	R/W	Dither Dither.DIFFUSION Dither.NOISE Dither.NONE Dither.PATTERN	type of dither
ditherAmount	R/W	Long	amount of dither. Only valid for diffusion (1 - 100)
forced	R/W	ForcedColors ForcedColors.BLACKWHITE ForcedColors.NONE ForcedColors.PRIMARIES ForcedColors.WEB	
matte	R/W	MatteType MatteType.BACKGROUND MatteType.BLACK MatteType.FOREGROUND MatteType.NETSCAPENONE MatteType.SEMIGRAY MatteType.WHITE	
palette	R/W	Palette Palette.EXACT Palette.LOCALADAPTIVE Palette.LOCALPERCEPTUAL Palette.LOCALSELECTIVE Palette.MACOSPALETTE Palette.MASTERADAPTIVE Palette.MASTERPERCEPTUAL Palette.MASTERSELECTIVE Palette.PREVIOUSPALETTE Palette.UNIFORM Palette.WEBPALETTE Palette.WINDOWSPALETTE	Type of palette (default: Palette.EXACT)
preserveExactColors	R/W	Boolean	
transparency	R/W	Boolean	

JPEGSaveOptions

Properties

Property	Access	Value Type	What it is
embedColorProfile	R/W	Boolean	embed color profile in document
formatOptions	R/W	FormatOptions FormatOptions.OPTIMIZEDBSELINE FormatOptions.PROGRESSIVE FormatOptions.STANDARDBASELINE	(default: FormatOptions.STANDARDBASELINE)
matte	R/W	MatteType MatteType.BACKGROUND MatteType.BLACK MatteType.FOREGROUND MatteType.NETSCAPENONE MatteType.SEMIGRAY MatteType.WHITE	
quality	R/W	Long	quality of produced image (0 - 12; default: 3)
scans	R/W	Long	number of scans. Only valid for progressive type JPEG files (3 - 5)

LabColor

Properties

Property	Access	Value Type	What it is
a	R/W	Double	the a-value (between -128.0 and 127.0)
b	R/W	Double	the b-value (between -128.0 and 127.0)
l	R/W	Double	the L-value (between 0.0 and 100.0)

LayerComp

Properties

Property	Access	Value Type	What it is
appearance	R/W	Boolean	use layer appearance
comment	R/W	ANYTHING	the description of the layer comp
name	R/W	String	the name of the layer comp
parent	RO	Object	the object's container
position	R/W	Boolean	use layer position
selected	RO	Boolean	the layer comp is currently selected
visibility	R/W	Boolean	use layer visibility

Methods

Method	What it does	Parameter Type	Returns
apply	Applies the layer comp to the document		
recapture	Recaptures the current layer state(s) for this layer comp		
remove	Deletes the object		
resetFromComp	Resets the layer comp state to the document state		

LayerComps

Properties

Property	Access	Value Type	What it is
length	RO	Long	number of elements in the collection
parent	RO	Object	the object's container

Methods

Method	What it does	Parameter Type	Returns
add	a layer comp	name as String comment as String appearance as Boolean position as Boolean visibility as Boolean	LayerComp
getByName	Get the first element in the collection with the provided name	name as String	LayerComp
removeAll			

Layers

Properties

Property	Access	Value Type	What it is
length	RO	Long	number of elements in the collection
parent	RO	Object	the object's container

Methods

Method	What it does	Parameter Type	Returns
getByName	Get the first element in the collection with the provided name	name as String	Layer
removeAll			

LayerSet

Properties

Property	Access	Value Type	What it is
allLocked	R/W	Boolean	
artLayers	RO	ArtLayers	the art layers in this layer set
blendMode	R/W	BlendMode BlendMode.COLORBLEND BlendMode.COLORBURN BlendMode.COLORDODGE BlendMode.DARKEN BlendMode.DIFFERENCE BlendMode.DISSOLVE BlendMode.EXCLUSION BlendMode.HARDLIGHT BlendMode.HUE BlendMode.LIGHTEN BlendMode.LINEARBURN BlendMode.LINEARDODGE BlendMode.LINEARLIGHT BlendMode.LUMINOSITY BlendMode.MULTIPLY BlendMode.NORMAL BlendMode.OVERLAY BlendMode.PASSTHROUGH BlendMode.PINLIGHT BlendMode.SATURATION BlendMode.SCREEN BlendMode.SOFTLIGHT BlendMode.VIVIDLIGHT	
bounds	RO	Array(UnitValue)	Bounding rectangle of the Layer
enabledChannels	R/W	Object	channels that are enabled for the layer set. Must be a list of component channels
layers	RO	Layers	the layers in this layer set
layerSets	RO	LayerSets	LayerSets contained within a LayerSet
linkedLayers	RO	Object	
name	R/W	String	the name of the layer
opacity	R/W	Double	master opacity of layer (0.0 - 100.0)
parent	RO	Object	the object's container
visible	R/W	Boolean	

Methods

Method	What it does	Parameter Type	Returns
duplicate	Creates a duplicate of the object	*relativeObject as Object* *insertionLocation as ElementPlacement* *ElementPlacement.INSIDE* *ElementPlacement.PLACEATBEGINNING* *ElementPlacement.PLACEATEND* *ElementPlacement.PLACEBEFORE* *ElementPlacement.PLACEAFTER*	Object (Layer)
link	Links the layer with another layer	with as Object (Layer)	
merge	Merges the layerset. Returns a reference to the art layer that is created by this method		ArtLayer
move	Moves the object	relativeObject as Object insertionLocation as ElementPlacement ElementPlacement.INSIDE ElementPlacement.PLACEATBEGINNING ElementPlacement.PLACEATEND ElementPlacement.PLACEBEFORE ElementPlacement.PLACEAFTER	
remove	Deletes the object		
resize		*horizontal as Double* *vertical as Double* *anchor as AnchorPosition* *AnchorPosition.BOTTOMCENTER* *AnchorPosition.BOTTOMLEFT* *AnchorPosition.BOTTOMRIGHT* *AnchorPosition.MIDDLECENTER* *AnchorPosition.MIDDLELEFT* *AnchorPosition.MIDDLERIGHT* *AnchorPosition.TOPCENTER* *AnchorPosition.TOPLEFT* *AnchorPosition.TOPRIGHT*	

Method	What it does	Parameter Type	Returns
rotate		angle as Double *anchor as AnchorPosition* *AnchorPosition.BOTTOMCENTER* *AnchorPosition.BOTTOMLEFT* *AnchorPosition.BOTTOMRIGHT* *AnchorPosition.MIDDLECENTER* *AnchorPosition.MIDDLELEFT* *AnchorPosition.MIDDLERIGHT* *AnchorPosition.TOPCENTER* *AnchorPosition.TOPLEFT* *AnchorPosition.TOPRIGHT*	
translate	Moves the position relative to its current position	*deltaX as UnitValue* *deltaY as UnitValue*	
unlink	Unlinks the layer set		

LayerSets

Properties

Property	Access	Value Type	What it is
length	RO	Long	number of elements in the collection
parent	RO	Object	the object's container

Methods

Method	What it does	Parameter Type	Returns
add	Creates a new object		LayerSet
getByName	Get the first element in the collection with the provided name	name as String	LayerSet
removeAll	Removes the layer set and any contained layers or layer sets		

Code (LayerSets.js)

```
$.level = 1;

// first close all the open documents
while (app.documents.length) {
    app.activeDocument.close();
}

// create a working document
var docRef = app.documents.add();

// create an array to hold the layer sets
var myLayerSets = new Array();

// a helpful array to hold some text for us
var textArray = Array("First", "Second", "Third");

// an indexer
var i = 0;

// create three layer sets at the top level
for (i = 0; i < 3; i++) {
    myLayerSets[i] = new Array();
```

```
    myLayerSets[i][0] = docRef.layerSets.add();
}

// rearrange them so the first one is on top, second next, etc.
myLayerSets[1][0].moveAfter(myLayerSets[0][0]);
myLayerSets[2][0].moveAfter(myLayerSets[1][0]);

// create a layer set inside each layer set
for (i = 0; i < 3; i++) {
    myLayerSets[i][0].name = textArray[i] + " Set";
    myLayerSets[i][1] = myLayerSets[i][0].layerSets.add();
    myLayerSets[i][1].name = "Inside " + textArray[i] + " Set";
}

// create another array to hold the layers
var myLayers = new Array();

// create a text layer with a description inside each layer set
for (i = 0; i < 3; i++) {
    myLayers[i] = myLayerSets[i][1].artLayers.add();
    myLayers[i].kind = LayerKind.TEXT;
    myLayers[i].textItem.contents = "Layer in " + textArray[i] + " Set Inside "
        + textArray[i] + " Set";
    myLayers[i].textItem.position = Array(app.activeDocument.width * i * 0.33,
        app.activeDocument.height * (i + 1) * 0.25);
    myLayers[i].textItem.size = 12;
}
```

PathItem

Properties

Property	Access	Value Type	What it is
kind	R/W	PathKind PathKind.CLIPPINGPATH PathKind.NORMALPATH PathKind.WORKPATH	
name			
parent	RO	PathKind PathKind.CLIPPINGPATH PathKind.NORMALPATH PathKind.WORKPATH	the object's container
subPathItems	RO	SubPathItems	sub items for this path item

Methods

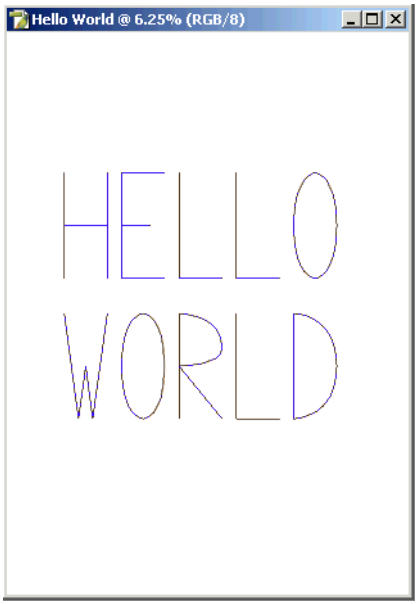
Method	What it does	Parameter Type	Returns
remove	Deletes this path		
duplicate	Duplicates this path with a new name	name as String	

Method	What it does	Parameter Type	Returns
fillPath	Fills the path with the following information	*fillColor as Anything* *mode as ColorBlendMode* *ColorBlendMode.BEHIND* *ColorBlendMode.CLEAR* *ColorBlendMode.COLOR* *ColorBlendMode.COLORBURN* *ColorBlendMode.COLORDODGE* *ColorBlendMode.DARKEN* *ColorBlendMode.DIFFERENCE* *ColorBlendMode.DISSOLVE* *ColorBlendMode.EXCLUSION* *ColorBlendMode.HARDLIGHT* *ColorBlendMode.HUE* *ColorBlendMode.LIGHTEN* *ColorBlendMode.LINEARBURN* *ColorBlendMode.LINEARDODGE* *ColorBlendMode.LINEARLIGHT* *ColorBlendMode.LUMINOSITY* *ColorBlendMode.MULTIPLY* *ColorBlendMode.NORMAL* *ColorBlendMode.OVERLAY* *ColorBlendMode.PINLIGHT* *ColorBlendMode.SATURATION* *ColorBlendMode.SCREEN* *ColorBlendMode.SOFTLIGHT* *ColorBlendMode.VIVIDLIGHT* *opacity as Double* *preserveTransparency as Boolean* *feather as Double* *wholePath as Boolean* *antiAlias as Boolean* *antiAlias.CRISP* *antiAlias.NONE* *antiAlias.SHARP* *antiAlias.SMOOTH* *antiAlias.STRONG*	
makeClippingPath	Makes this path item the clipping path for this document	*flatness as Double*	

Method	What it does	Parameter Type	Returns
makeSelection	Makes a selection from this path	*feather as Double* *antiAlias as Boolean* *antiAlias as Boolean* *antiAlias.CRISP* *antiAlias.NONE* *antiAlias.SHARP* *antiAlias.SMOOTH* *antiAlias.STRONG* *operation as SelectionType* *SelectionType.DIMINISH* *SelectionType.EXTEND* *SelectionType.INTERSECT* *SelectionType.REPLACE*	
strokePath	Strokes the path with the following information	*tool as ToolType* *ToolType.ARTHISTORYBRUSH* *ToolType.BACKGROUNDERASER* *ToolType.BLUR* *ToolType.BRUSH* *ToolType.BURN* *ToolType.CLONESTAMP* *ToolType.COLORREPLACEMENTTOOL* *ToolType.DODGE* *ToolType.ERASER* *ToolType.HEALINGBRUSH* *ToolType.HISTORYBRUSH* *ToolType.PATTERNSTAMP* *ToolType.PENCIL* *ToolType.SHARPEN* *ToolType.SMUDGE* *ToolType.SPONGE* *simulatePressure as Boolean*	

Sample Script

The following script manipulates multiple art paths to produce a multi-colored version of "Hello World".



Code (ArtPaths.js)

```
// Save the current preferences
var startRulerUnits = app.preferences.rulerUnits;
var startTypeUnits = app.preferences.typeUnits;
var startDisplayDialogs = app.displayDialogs;

// Set Photoshop to use pixels and display no dialogs
app.preferences.rulerUnits = Units.PIXELS;
app.preferences.typeUnits = TypeUnits.PIXELS;
app.displayDialogs = DialogModes.NO;

// first close all the open documents
while (app.documents.length) {
   app.activeDocument.close();
}

// create a document to work with
var docRef = app.documents.add(5000, 7000, 72, "Hello World");

// figure out how big a letter would be
```

```
var letterBoxWidth = docRef.width / 7;
var letterWidth = letterBoxWidth * .75;
var letterBoxHeight = docRef.height / 4;
var letterHeight = letterBoxHeight * .75;

// move to the top left corner for the first letter
var letterLocationX =  letterBoxWidth.value;
var letterLocationY = letterBoxHeight.value;

// this array will hold all the sub paths
// each AddLetter.Path routine will append to the end
var letterSubPaths = new Array();

// add all the paths needed for the letter H
AddLetterHPath(letterSubPaths, letterLocationX, letterLocationY,
letterWidth.value, letterHeight.value);

// move over to the next letter
letterLocationX += letterBoxWidth.value;
AddLetterEPath(letterSubPaths, letterLocationX, letterLocationY,
letterWidth.value, letterHeight.value);

letterLocationX += letterBoxWidth.value;
AddLetterLPath(letterSubPaths, letterLocationX, letterLocationY,
letterWidth.value, letterHeight.value);

letterLocationX += letterBoxWidth.value;
AddLetterLPath(letterSubPaths, letterLocationX, letterLocationY,
letterWidth.value, letterHeight.value);

letterLocationX += letterBoxWidth.value;
AddLetterOPath(letterSubPaths, letterLocationX, letterLocationY,
letterWidth.value, letterHeight.value);

// move back to the left and down one row
letterLocationX = letterBoxWidth.value;
letterLocationY += letterBoxHeight.value;
AddLetterWPath(letterSubPaths, letterLocationX, letterLocationY,
letterWidth.value, letterHeight.value);

letterLocationX += letterBoxWidth.value;
AddLetterOPath(letterSubPaths, letterLocationX, letterLocationY,
letterWidth.value, letterHeight.value);

letterLocationX += letterBoxWidth.value;
AddLetterRPath(letterSubPaths, letterLocationX, letterLocationY,
letterWidth.value, letterHeight.value);

letterLocationX += letterBoxWidth.value;
```

```
AddLetterLPath(letterSubPaths, letterLocationX, letterLocationY,
letterWidth.value, letterHeight.value);

letterLocationX += letterBoxWidth.value;
AddLetterDPath(letterSubPaths, letterLocationX, letterLocationY,
letterWidth.value, letterHeight.value);

// create the path
var myPathItem = docRef.pathItems.add("Testing", letterSubPaths);

// stroke it so we can see something
myPathItem.strokePath(ToolType.BRUSH);

// deselect it
myPathItem.deselect();


// each of the functions below are for each individual letter
// bad design but it makes the stuff above easier to read

function AddLetterDPath(inOutSubPaths, inX, inY, inWidth, inHeight) {
    // create the letter D
    var letterPoints = new Array();

    letterPoints[0] = new PathPointInfo;
    letterPoints[0].kind = PointKind.CORNERPOINT;
    letterPoints[0].anchor = Array(inX, inY);
    letterPoints[0].leftDirection = letterPoints[0].anchor;
    letterPoints[0].rightDirection = letterPoints[0].anchor;

    letterPoints[1] = new PathPointInfo;
    letterPoints[1].kind = PointKind.CORNERPOINT;
    letterPoints[1].anchor = Array(inX + inWidth, inY + inHeight / 2);
    letterPoints[1].leftDirection = Array(inX + inWidth, inY + inHeight);
    letterPoints[1].rightDirection = Array(inX + inWidth, inY);

    letterPoints[2] = new PathPointInfo;
    letterPoints[2].kind = PointKind.CORNERPOINT;
    letterPoints[2].anchor = Array(inX, inY + inHeight);
    letterPoints[2].leftDirection = letterPoints[2].anchor;
    letterPoints[2].rightDirection = letterPoints[2].anchor;

    letterPoints[3] = new PathPointInfo;
    letterPoints[3].kind = PointKind.CORNERPOINT;
    letterPoints[3].anchor = Array(inX, inY);
    letterPoints[3].leftDirection = letterPoints[3].anchor;
    letterPoints[3].rightDirection = letterPoints[3].anchor;


    var insertIndex = inOutSubPaths.length;
```

```
    inOutSubPaths[insertIndex] = new SubPathInfo();
    inOutSubPaths[insertIndex].operation = ShapeOperation.SHAPEXOR;
    inOutSubPaths[insertIndex].closed = false;
    inOutSubPaths[insertIndex].entireSubPath = letterPoints;
}


function AddLetterRPath(inOutSubPaths, inX, inY, inWidth, inHeight) {
    // create the letter R
    var letterPoints = new Array();

    letterPoints[0] = new PathPointInfo;
    letterPoints[0].kind = PointKind.CORNERPOINT;
    letterPoints[0].anchor = Array(inX, inY + inHeight);
    letterPoints[0].leftDirection = letterPoints[0].anchor;
    letterPoints[0].rightDirection = letterPoints[0].anchor;

    letterPoints[1] = new PathPointInfo;
    letterPoints[1].kind = PointKind.CORNERPOINT;
    letterPoints[1].anchor = Array(inX, inY);
    letterPoints[1].leftDirection = letterPoints[1].anchor;
    letterPoints[1].rightDirection = letterPoints[1].anchor;

    letterPoints[2] = new PathPointInfo;
    letterPoints[2].kind = PointKind.CORNERPOINT;
    letterPoints[2].anchor = Array(inX + inWidth, inY + inHeight * .33);
    letterPoints[2].leftDirection = Array(inX + inWidth, inY + inHeight / 2);
    letterPoints[2].rightDirection = Array(inX + inWidth, inY);

    letterPoints[3] = new PathPointInfo;
    letterPoints[3].kind = PointKind.CORNERPOINT;
    letterPoints[3].anchor = Array(inX, inY + inHeight / 2);
    letterPoints[3].leftDirection = letterPoints[3].anchor;
    letterPoints[3].rightDirection = letterPoints[3].anchor;

    letterPoints[4] = new PathPointInfo;
    letterPoints[4].kind = PointKind.CORNERPOINT;
    letterPoints[4].anchor = Array(inX + inWidth, inY + inHeight);
    letterPoints[4].leftDirection = letterPoints[4].anchor;
    letterPoints[4].rightDirection = letterPoints[4].anchor;

    var insertIndex = inOutSubPaths.length;

    inOutSubPaths[insertIndex] = new SubPathInfo();
    inOutSubPaths[insertIndex].operation = ShapeOperation.SHAPEXOR;
    inOutSubPaths[insertIndex].closed = false;
    inOutSubPaths[insertIndex].entireSubPath = letterPoints;
}
```

```
function AddLetterWPath(inOutSubPaths, inX, inY, inWidth, inHeight) {
    // create the letter W
    var letterPoints = new Array();

    letterPoints[0] = new PathPointInfo;
    letterPoints[0].kind = PointKind.CORNERPOINT;
    letterPoints[0].anchor = Array(inX, inY);
    letterPoints[0].leftDirection = letterPoints[0].anchor;
    letterPoints[0].rightDirection = letterPoints[0].anchor;

    letterPoints[1] = new PathPointInfo;
    letterPoints[1].kind = PointKind.CORNERPOINT;
    letterPoints[1].anchor = Array(inX + inWidth * .33, inY + inHeight);
    letterPoints[1].leftDirection = letterPoints[1].anchor;
    letterPoints[1].rightDirection = letterPoints[1].anchor;

    letterPoints[2] = new PathPointInfo;
    letterPoints[2].kind = PointKind.CORNERPOINT;
    letterPoints[2].anchor = Array(inX + inWidth / 2, inY + inHeight / 2);
    letterPoints[2].leftDirection = letterPoints[2].anchor;
    letterPoints[2].rightDirection = letterPoints[2].anchor;

    letterPoints[3] = new PathPointInfo;
    letterPoints[3].kind = PointKind.CORNERPOINT;
    letterPoints[3].anchor = Array(inX + inWidth * .66, inY + inHeight);
    letterPoints[3].leftDirection = letterPoints[3].anchor;
    letterPoints[3].rightDirection = letterPoints[3].anchor;

    letterPoints[4] = new PathPointInfo;
    letterPoints[4].kind = PointKind.CORNERPOINT;
    letterPoints[4].anchor = Array(inX + inWidth, inY);
    letterPoints[4].leftDirection = letterPoints[4].anchor;
    letterPoints[4].rightDirection = letterPoints[4].anchor;

    var insertIndex = inOutSubPaths.length;

    inOutSubPaths[insertIndex] = new SubPathInfo();
    inOutSubPaths[insertIndex].operation = ShapeOperation.SHAPEXOR;
    inOutSubPaths[insertIndex].closed = false;
    inOutSubPaths[insertIndex].entireSubPath = letterPoints;
}


function AddLetterOPath(inOutSubPaths, inX, inY, inWidth, inHeight) {

    // create the letter O
    var letterPoints = new Array();

    letterPoints[0] = new PathPointInfo;
```

```
    letterPoints[0].kind = PointKind.CORNERPOINT;
    letterPoints[0].anchor = Array(inX + inWidth / 2, inY);
    letterPoints[0].leftDirection = letterPoints[0].anchor;
    letterPoints[0].rightDirection = letterPoints[0].anchor;

    letterPoints[1] = new PathPointInfo;
    letterPoints[1].kind = PointKind.CORNERPOINT;
    letterPoints[1].anchor = Array(inX + inWidth, inY + inHeight / 2);
    letterPoints[1].leftDirection = Array(inX + inWidth, inY + inHeight);
    letterPoints[1].rightDirection = Array(inX + inWidth, inY);

    letterPoints[2] = new PathPointInfo;
    letterPoints[2].kind = PointKind.CORNERPOINT;
    letterPoints[2].anchor = Array(inX + inWidth / 2, inY + inHeight);
    letterPoints[2].leftDirection = letterPoints[2].anchor;
    letterPoints[2].rightDirection = letterPoints[2].anchor;

    letterPoints[3] = new PathPointInfo;
    letterPoints[3].kind = PointKind.CORNERPOINT;
    letterPoints[3].anchor = Array(inX, inY + inHeight / 2);
    letterPoints[3].leftDirection = Array(inX, inY);
    letterPoints[3].rightDirection = Array(inX, inY + inHeight);

    letterPoints[4] = new PathPointInfo;
    letterPoints[4].kind = PointKind.CORNERPOINT;
    letterPoints[4].anchor = Array(inX + inWidth / 2, inY);
    letterPoints[4].leftDirection = letterPoints[4].anchor;
    letterPoints[4].rightDirection = letterPoints[4].anchor;

    var insertIndex = inOutSubPaths.length;

    inOutSubPaths[insertIndex] = new SubPathInfo();
    inOutSubPaths[insertIndex].operation = ShapeOperation.SHAPEXOR;
    inOutSubPaths[insertIndex].closed = false;
    inOutSubPaths[insertIndex].entireSubPath = letterPoints;
}


function AddLetterLPath(inOutSubPaths, inX, inY, inWidth, inHeight) {

    // create the letter L
    var letterPoints = new Array();

    letterPoints[0] = new PathPointInfo;
    letterPoints[0].kind = PointKind.CORNERPOINT;
    letterPoints[0].anchor = Array(inX, inY);
    letterPoints[0].leftDirection = letterPoints[0].anchor;
    letterPoints[0].rightDirection = letterPoints[0].anchor;

    letterPoints[1] = new PathPointInfo;
```

```
    letterPoints[1].kind = PointKind.CORNERPOINT;
    letterPoints[1].anchor = Array(inX, inY + inHeight);
    letterPoints[1].leftDirection = letterPoints[1].anchor;
    letterPoints[1].rightDirection = letterPoints[1].anchor;

    letterPoints[2] = new PathPointInfo;
    letterPoints[2].kind = PointKind.CORNERPOINT;
    letterPoints[2].anchor = Array(inX + inWidth, inY + inHeight);
    letterPoints[2].leftDirection = letterPoints[2].anchor;
    letterPoints[2].rightDirection = letterPoints[2].anchor;

    var insertIndex = inOutSubPaths.length;

    inOutSubPaths[insertIndex] = new SubPathInfo();
    inOutSubPaths[insertIndex].operation = ShapeOperation.SHAPEXOR;
    inOutSubPaths[insertIndex].closed = false;
    inOutSubPaths[insertIndex].entireSubPath = letterPoints;
}


function AddLetterEPath(inOutSubPaths, inX, inY, inWidth, inHeight) {

    // create the letter E top, left, and bottom side
    var letterPoints = new Array();

    letterPoints[0] = new PathPointInfo;
    letterPoints[0].kind = PointKind.CORNERPOINT;
    letterPoints[0].anchor = Array(inX + inWidth, inY);
    letterPoints[0].leftDirection = letterPoints[0].anchor;
    letterPoints[0].rightDirection = letterPoints[0].anchor;

    letterPoints[1] = new PathPointInfo;
    letterPoints[1].kind = PointKind.CORNERPOINT;
    letterPoints[1].anchor = Array(inX, inY);
    letterPoints[1].leftDirection = letterPoints[1].anchor;
    letterPoints[1].rightDirection = letterPoints[1].anchor;

    letterPoints[2] = new PathPointInfo;
    letterPoints[2].kind = PointKind.CORNERPOINT;
    letterPoints[2].anchor = Array(inX, inY + inHeight);
    letterPoints[2].leftDirection = letterPoints[2].anchor;
    letterPoints[2].rightDirection = letterPoints[2].anchor;

    letterPoints[3] = new PathPointInfo;
    letterPoints[3].kind = PointKind.CORNERPOINT;
    letterPoints[3].anchor = Array(inX + inWidth, inY + inHeight);
    letterPoints[3].leftDirection = letterPoints[3].anchor;
    letterPoints[3].rightDirection = letterPoints[3].anchor;

    var insertIndex = inOutSubPaths.length;
```

```
    inOutSubPaths[insertIndex] = new SubPathInfo();
    inOutSubPaths[insertIndex].operation = ShapeOperation.SHAPEXOR;
    inOutSubPaths[insertIndex].closed = false;
    inOutSubPaths[insertIndex].entireSubPath = letterPoints;

    // create the letter E cross bar
    var letterPoints = new Array();

    letterPoints[0] = new PathPointInfo;
    letterPoints[0].kind = PointKind.CORNERPOINT;
    letterPoints[0].anchor = Array(inX, inY + inHeight / 2);
    letterPoints[0].leftDirection = letterPoints[0].anchor;
    letterPoints[0].rightDirection = letterPoints[0].anchor;

    letterPoints[1] = new PathPointInfo;
    letterPoints[1].kind = PointKind.CORNERPOINT;
    letterPoints[1].anchor = Array(inX + inWidth * 0.66, inY + inHeight / 2);
    letterPoints[1].leftDirection = letterPoints[1].anchor;
    letterPoints[1].rightDirection = letterPoints[1].anchor;

    insertIndex = inOutSubPaths.length;

    inOutSubPaths[insertIndex] = new SubPathInfo();
    inOutSubPaths[insertIndex].operation = ShapeOperation.SHAPEXOR;
    inOutSubPaths[insertIndex].closed = false;
    inOutSubPaths[insertIndex].entireSubPath = letterPoints;
}


function AddLetterHPath(inOutSubPaths, inX, inY, inWidth, inHeight) {

    // create the letter H left side
    var letterPoints = new Array();

    letterPoints[0] = new PathPointInfo;
    letterPoints[0].kind = PointKind.CORNERPOINT;
    letterPoints[0].anchor = Array(inX, inY);
    letterPoints[0].leftDirection = letterPoints[0].anchor;
    letterPoints[0].rightDirection = letterPoints[0].anchor;

    letterPoints[1] = new PathPointInfo;
    letterPoints[1].kind = PointKind.CORNERPOINT;
    letterPoints[1].anchor = Array(inX, inY + inHeight);
    letterPoints[1].leftDirection = letterPoints[1].anchor;
    letterPoints[1].rightDirection = letterPoints[1].anchor;

    var insertIndex = inOutSubPaths.length;

    inOutSubPaths[insertIndex] = new SubPathInfo();
```

```
    inOutSubPaths[insertIndex].operation = ShapeOperation.SHAPEXOR;
    inOutSubPaths[insertIndex].closed = false;
    inOutSubPaths[insertIndex].entireSubPath = letterPoints;

    // create the letter H cross bar
    var letterPoints = new Array();

    letterPoints[0] = new PathPointInfo;
    letterPoints[0].kind = PointKind.CORNERPOINT;
    letterPoints[0].anchor = Array(inX, inY + inHeight / 2);
    letterPoints[0].leftDirection = letterPoints[0].anchor;
    letterPoints[0].rightDirection = letterPoints[0].anchor;

    letterPoints[1] = new PathPointInfo;
    letterPoints[1].kind = PointKind.CORNERPOINT;
    letterPoints[1].anchor = Array(inX + inWidth, inY + inHeight / 2);
    letterPoints[1].leftDirection = letterPoints[1].anchor;
    letterPoints[1].rightDirection = letterPoints[1].anchor;

    insertIndex = inOutSubPaths.length;

    inOutSubPaths[insertIndex] = new SubPathInfo();
    inOutSubPaths[insertIndex].operation = ShapeOperation.SHAPEXOR;
    inOutSubPaths[insertIndex].closed = false;
    inOutSubPaths[insertIndex].entireSubPath = letterPoints;

    // create the letter H right side
    var letterPoints = new Array();

    letterPoints[0] = new PathPointInfo;
    letterPoints[0].kind = PointKind.CORNERPOINT;
    letterPoints[0].anchor = Array(inX + inWidth, inY);
    letterPoints[0].leftDirection = letterPoints[0].anchor;
    letterPoints[0].rightDirection = letterPoints[0].anchor;

    letterPoints[1] = new PathPointInfo;
    letterPoints[1].kind = PointKind.CORNERPOINT;
    letterPoints[1].anchor = Array(inX + inWidth, inY + inHeight);
    letterPoints[1].leftDirection = letterPoints[1].anchor;
    letterPoints[1].rightDirection = letterPoints[1].anchor;

    insertIndex = inOutSubPaths.length;

    inOutSubPaths[insertIndex] = new SubPathInfo();
    inOutSubPaths[insertIndex].operation = ShapeOperation.SHAPEXOR;
    inOutSubPaths[insertIndex].closed = false;
    inOutSubPaths[insertIndex].entireSubPath = letterPoints;
}
```

```
// Reset the application preferences
app.preferences.rulerUnits = startRulerUnits;
app.preferences.typeUnits = startTypeUnits;
app.displayDialogs = startDisplayDialogs;
```

PathItems

Properties

Property	Access	Value Type	What it is
length	RO	Long	number of elements in the collection
parent	RO	Object	the object's container

Methods

Method	What it does	Parameter Type	Returns
add	Creates a new path item	name as String entirePath as Object	PathItem
getByName	Get the first element in the collection with the provided name	name as String	PathItem
removeAll			

PathPoint

Properties

Property	Access	Value Type	What it is
anchor	R/W	Array(UnitValue)	the edit point on the curve -- leftDirection/rightDirection are points representing the control handle end points
kind	R/W	PointKind PointKind.CORNERPOINT PointKind.SMOOTHPOINT	
leftDirection	R/W	Array(UnitValue)	
parent	RO	Object	the object's container
rightDirection	R/W	Array(UnitValue)	

PathPointInfo

Properties

Property	Access	Value Type	What it is
anchor	R/W	ANYTHING	the position of the anchor (in coordinates)
kind	R/W	PointKind PointKind.CORNERPOINT PointKind.SMOOTHPOINT	the point type, smooth/conner
leftDirection	R/W	ANYTHING	location of the left direction point (in position)
rightDirection	R/W	ANYTHING	location of the left direction point (out position)

PathPoints

Properties

Property	Access	Value Type	What it is
length	RO	Long	number of elements in the collection
parent	RO	Object	the object's container

PDFOpenOptions

Properties

Property	Access	Value Type	What it is
antiAlias	R/W	Boolean	use antialias?
constrainProportions	R/W	Boolean	constrain proportions of image
height	R/W	UnitValue	height of image (unit value)
mode	R/W	OpenDocumentMode OpenDocumentMode.CMYK OpenDocumentMode.GRAYSC OpenDocumentMode.ALE OpenDocumentMode.LAB OpenDocumentMode.RGB	the document mode
page	R/W	Long	number of page to open
resolution	R/W	Double	the resolution of the document (in pixels per inch)
width	R/W	UnitValue	width of image (unit value)

PDFSaveOptions

Properties

Property	Access	Value Type	What it is
alphaChannels	R/W	Boolean	save alpha channels
annotations	R/W	Boolean	save annotations
downgradeColorProfile	R/W	Boolean	should the embedded color profile be downgraded to version 2
embedColorProfile	R/W	Boolean	embed color profile in document
embedFonts	R/W	Boolean	embed fonts? Only valid if vector data is included
encoding	R/W	PDFEncoding PDFEncoding.JPEG PDFEncoding.PDFZIP	(default: PDFEncoding.PDFZIP)
interpolation	R/W	Boolean	use image interpolation?
jpegQuality	R/W	Long	quality of produced image. Only valid for JPEG encoded PDF documents (0 - 12)
layers	R/W	Boolean	save layers
spotColors	R/W	Boolean	save spot colors
transparency	R/W	Boolean	
useOutlines	R/W	Boolean	use outlines for text? Only valid if vector data is included
vectorData	R/W	Boolean	include vector data

PhotoCDOpenOptions

Properties

Property	Access	Value Type	What it is
colorProfileName	R/W	String	profile to use when reading the image
colorSpace	R/W	PhotoCDColorSpace PhotoCDColorSpace.LAB16 PhotoCDColorSpace.LAB8 PhotoCDColorSpace.RGB16 PhotoCDColorSpace.RGB8	colorspace for image
orientation	R/W	Orientation Orientation.LANDSCAPE Orientation.PORTRAIT	
pixelSize	R/W	PhotoCDSize PhotoCDSize.EXTRALARGE PhotoCDSize.LARGE PhotoCDSize.MAXIMUM PhotoCDSize.MEDIUM PhotoCDSize.MINIMUM PhotoCDSize.SMALL	dimensions of image
resolution	R/W	Double	the resolution of the image (in pixels per inch)

PhotoshopSaveOptions

Properties

Property	Access	Value Type	What it is
alphaChannels	R/W	Boolean	save alpha channels
annotations	R/W	Boolean	save annotations
embedColorProfile	R/W	Boolean	embed color profile in document
layers	R/W	Boolean	save layers
spotColors	R/W	Boolean	save spot colors

PICTFileSaveOptions

Properties

Property	Access	Value Type	What it is
alphaChannels	R/W	Boolean	save alpha channels
compression	R/W	PICTCompression PICTCompression.JPEGHIGHPICT PICTCompression.JPEGLOWPICT PICTCompression.JPEGMAXIMUMPICT PICTCompression.JPEGMEDIUMPICT PICTCompression.NONE	(default: PICTCompression.NONE)
embedColorProfile	R/W	Boolean	embed color profile in document
resolution	R/W	PICTBitsPerPixels PICTBitsPerPixels.EIGHT PICTBitsPerPixels.FOUR PICTBitsPerPixels.SIXTEEN PICTBitsPerPixels.THIRTYTWO PICTBitsPerPixels.TWO	number of bits per pixel

PICTResourceSaveOptions

Properties

Property	Access	Value Type	What it is
alphaChannels	R/W	Boolean	save alpha channels
compression	R/W	PICTCompression PICTCompression.JPEGHIGHPICT PICTCompression.JPEGLOWPICT PICTCompression.JPEGMAXIMUMPICT PICTCompression.JPEGMEDIUMPICT PICTCompression.NONE	(default: PICTCompression.NONE)
embedColorProfile	R/W	Boolean	embed color profile in document
name	R/W	String	name of PICT resource (default: \"\")
resolution	R/W	PICTBitsPerPixels PICTBitsPerPixels.EIGHT PICTBitsPerPixels.FOUR PICTBitsPerPixels.SIXTEEN PICTBitsPerPixels.THIRTYTWO PICTBitsPerPixels.TWO	number of bits per pixel
resourceID	R/W	PICTBitsPerPixels PICTBitsPerPixels.EIGHT PICTBitsPerPixels.FOUR PICTBitsPerPixels.SIXTEEN PICTBitsPerPixels.THIRTYTWO PICTBitsPerPixels.TWO	ID of PICT resource (default: 128)

PixarSaveOptions

Properties

Property	Access	Value Type	What it is
alphaChannels	R/W	Boolean	save alpha channels

PNGSaveOptions

Properties

Property	Access	Value Type	What it is
interlaced	R/W	Boolean	should rows be interlaced? (default: false)

Preferences

Properties

Property	Access	Value Type	What it is
additionalPluginFolder	R/W	File	
appendExtension	R/W	SaveBehavior SaveBehavior.ALWAYSSAVE SaveBehavior.ASKWHENSAVING SaveBehavior.NEVERSAVE	on Windows, files are always saved with extensions.
askBeforeSavingLayeredTIFF	R/W	Boolean	
autoUpdateOpenDocuments	R/W	Boolean	
beepWhenDone	R/W	Boolean	
colorChannelsInColor	R/W	Boolean	
colorPicker	R/W	ColorPicker ColorPicker.ADOBE ColorPicker.APPLE ColorPicker.PLUGIN ColorPicker.WINDOWS	
columnGutter	R/W	Double	gutter of columns (in points)
columnWidth	R/W	Double	width of columns (in points)
createFirstSnapshot	R/W	Boolean	automatically make first snapshot when a new document is created?
dynamicColorSliders	R/W	Boolean	
editLogItems	R/W	EditLogItemsType EditLogItemsType.CONCISE EditLogItemsType.DETAILED EditLogItemsType.SESSIONONLY	options for edit log items
exportClipboard	R/W	Boolean	
fullSizePreview	R/W	Boolean	Mac only

Property	Access	Value Type	What it is
gamutWarningOpacity	R/W	Double	
gridSize	R/W	GridSize GridSize.LARGE GridSize.MEDIUM GridSize.NONE GridSize.SMALL	
gridStyle	R/W	GridLineStyle GridLineStyle.DASHED GridLineStyle.DOTTED GridLineStyle.SOLID	
gridSubDivisions	R/W	Long	
guideStyle	R/W	GuideLineStyle GuideLineStyle.DASHED GuideLineStyle.SOLID	
iconPreview	R/W	Boolean	Mac only
imageCacheForHistograms	R/W	Boolean	
imageCacheLevels	R/W	Long	
imagePreviews	R/W	SaveBehavior SaveBehavior.ALWAYSSAVE SaveBehavior.ASKWHENSAVING SaveBehavior.NEVERSAVE	
interpolation	R/W	ResampleMethod *ResampleMethod.BICUBIC* *ResampleMethod.BICUBICSHARPER* *ResampleMethod.BICUBICSMOOTHER* *ResampleMethod.BILINEAR* *ResampleMethod.NEARESTNEIGHBOR* *ResampleMethod.NONE*	
keyboardZoomResizesWindows	R/W	Boolean	
macOSThumbnail	R/W	Boolean	Mac only
maximizeCompatibility	R/W	QueryStateType QueryStateType.ALWAYS QueryStateType.ASK QueryStateType.NEVER	maximize compatibility for Photoshop (PSD) files
maxRAMuse	R/W	Long	Maximum percentage of available RAM used by Photoshop (5 - 100)

Property	Access	Value Type	What it is
nonLinearHistory	R/W	Boolean	allow non-linear history?
numberOfHistoryStates	R/W	Long	number of history states to remember (between 1 and 100)
otherCursors	R/W	OtherPaintingCursors OtherPaintingCursors.PRECISEOTHER OtherPaintingCursors.STANDARDOTHER	
paintingCursors	R/W	PaintingCursors PaintingCursors.BRUSHSIZE PaintingCursors.PRECISE PaintingCursors.STANDARD	
parent	RO	Object	the object's container
pixelDoubling	R/W	Boolean	
pointSize	R/W	PointType PointType.POSTSCRIPT PointType.TRADITIONAL	size of point/pica
recentFileListLength	R/W	Long	number of items in the recent file list (between 0 and 30)
rulerUnits	R/W	Units Units.CM Units.INCHES Units.MM Units.PERCENT Units.PICAS Units.PIXELS Units.POINTS	Note: this is the unit that the scripting system will use when receiving and returning values
saveLogItems	R/W	SaveLogItemsType SaveLogItemsType.LOGFILE SaveLogItemsType.LOGFILEANDMETADATA SaveLogItemsType.METADATA	options for saving the history items
saveLogItemsFile	R/W	File	

Property	Access	Value Type	What it is
savePaletteLocations	R/W	Boolean	
showAsianTextOptions	R/W	Boolean	
showEnglishFontNames	R/W	Boolean	
showSliceNumber	R/W	Boolean	
showToolTips	R/W	Boolean	
smartQuotes	R/W	Boolean	
typeUnits	R/W	TypeUnits TypeUnits.MM TypeUnits.PIXELS TypeUnits.POINTS	unit type-size that the numeric inputs are assumed to represent
useAdditionalPluginFolder	R/W	Boolean	
useDiffusionDither	R/W	Boolean	
useHistoryLog	R/W	Boolean	
useLowerCaseExtension	R/W	Boolean	should the file extension be lowercase
useShiftKeyForToolSwitch	R/W	Boolean	
useVideoAlpha	R/W	Boolean	this option requires hardware support
windowsTumbnail	R/W	Boolean	this option requires hardware support

PresentationOptions

Properties

Property	Access	Value Type	What it is
autoAdvance	R/W	Boolean	auto advance when viewing (default: true)
downgradeColorProfile	R/W	Boolean	should the embedded color profile be downgraded to version 2 (default: false)
embedFonts	R/W	Boolean	embed fonts? Only valid if a text layer is included (default: false)
encoding	R/W	PDFEncoding	(default: PDFEncoding.PDFZIP)
interpolation	R/W	Boolean	use image interpolation? (default: false)
interval	R/W	Long	time in seconds before auto advancing the view (default: 5)
jpegQuality	R/W	Long	quality of produced image. Only valid for JPEG encoded PDF documents (0 12; default: 10)
loop	R/W	Boolean	loop after last page (default: false)
presentation	R/W	Boolean	true if the file type is presentation false for Multi-Page document (default: false)
transition	R/W	TransitionType	transition type when switching to the next document (default: Transition-Type.NONE)
transparency	R/W	Boolean	(default: true)
vectorData	R/W	Boolean	include vector data (default: false)
view	R/W	Boolean	view the document after saving (default: false)

RawFormatOpenOptions

Properties

Property	Access	Value Type	What it is
bitsPerChannel	R/W	Long	number of bits for each channel (8 or 16)
byteOrder	R/W	Byte Order	only relevant for images with 16 bits per channel
channelNumber	R/W	Long	number of channels in image
headerSize	R/W	Long	
height	R/W	Long	height of image (in pixels)
interleaveChannels	R/W	Boolean	are the channels in the image interleaved?
retainHeader	R/W	Boolean	retain header when saving?
width	R/W	Long	width of image (in pixels)

RawSaveOptions

Properties

Property	Access	Value Type	What it is
alphaChannels	R/W	Boolean	save alpha channels
spotColors	R/W	Boolean	save spot colors

RGBColor

Properties

Property	Access	Value Type	What it is
blue	R/W	Double	the blue color value (0.0 - 255.0; default: 255.0)
green	R/W	Double	the green color value (0.0 - 255.0; default: 255.0)
hexValue	R/W	String	Hex representation of this color
red	R/W	Double	Hex representation of this color

Selection

Properties

Property	Access	Value Type	What it is
parent	RO	Object	the object's container

Methods

Method	What it does	Parameter Type	Returns
clear	Clears selection		
contract	Contracts the selection	by UnitValue	
copy	Copies selection to the clipboard	*merge as Boolean*	
cut	Cuts current selection to the clipboard		
deselect			
expand	Expands selection	by as UnitValue	
feather	Feathers edges of selection	by as UnitValue	

Method	What it does	Parameter Type	Returns
fill	Fills the selection	fillType as ANYTHING *mode as ColorBlendMode* *ColorBlendMode.BEHIND* *ColorBlendMode.CLEAR* *ColorBlendMode.COLOR* *ColorBlendMode.COLORBURN* *ColorBlendMode.COLORDODGE* *ColorBlendMode.DARKEN* *ColorBlendMode.DIFFERENCE* *ColorBlendMode.DISSOLVE* *ColorBlendMode.EXCLUSION* *ColorBlendMode.HARDLIGHT* *ColorBlendMode.HUE* *ColorBlendMode.LIGHTEN* *ColorBlendMode.LINEARBURN* *ColorBlendMode.LINEARDODGE* *ColorBlendMode.LINEARLIGHT* *ColorBlendMode.LUMINOSITY* *ColorBlendMode.MULTIPLY* *ColorBlendMode.NORMAL* *ColorBlendMode.OVERLAY* *ColorBlendMode.PINLIGHT* *ColorBlendMode.SATURATION* *ColorBlendMode.SCREEN* *ColorBlendMode.SOFTLIGHT* *ColorBlendMode.VIVIDLIGHT* *opacity as Long* *preserveTransparency as Boolean*	
grow	Grows selection to include all adjacent pixels falling within the specified tolerance range	tolerance as Long antiAlias as Boolean *antiAlias.CRISP* *antiAlias.NONE* *antiAlias.SHARP* *antiAlias.SMOOTH* *antiAlias.STRONG*	
invert	Inverts the selection		

Method	What it does	Parameter Type	Returns
load	Loads the selection from a channel	from as Channel *combination as SelectionType* *SelectionType.DIMINISH* *SelectionType.EXTEND* *SelectionType.INTERSECT* *SelectionType.REPLACE* *inverting as Boolean*	
resize		*horizontal as Double* *vertical as Double* *anchor as AnchorPosition* *AnchorPosition.BOTTOMCENTER* *.AnchorPosition.BOTTOMLEFT* *AnchorPosition.BOTTOMRIGHT* *AnchorPosition.MIDDLECENTER* *AnchorPosition.MIDDLELEFT* *AnchorPosition.MIDDLERIGHT* *AnchorPosition.TOPCENTER* *AnchorPosition.TOPLEFT* *AnchorPosition.TOPRIGHT*	
resizeBoundary	Scales the boundary of selection	*horizontal as Double* *vertical as Double* *anchor as AnchorPosition* *AnchorPosition.BOTTOMCENTER* *.AnchorPosition.BOTTOMLEFT* *AnchorPosition.BOTTOMRIGHT* *AnchorPosition.MIDDLECENTER* *AnchorPosition.MIDDLELEFT* *AnchorPosition.MIDDLERIGHT* *AnchorPosition.TOPCENTER* *AnchorPosition.TOPLEFT* *AnchorPosition.TOPRIGHT*	
rotate		*angle as Double* *anchor as AnchorPosition* *AnchorPosition.BOTTOMCENTER* *.AnchorPosition.BOTTOMLEFT* *AnchorPosition.BOTTOMRIGHT* *AnchorPosition.MIDDLECENTER* *AnchorPosition.MIDDLELEFT* *AnchorPosition.MIDDLERIGHT* *AnchorPosition.TOPCENTER* *AnchorPosition.TOPLEFT* *AnchorPosition.TOPRIGHT*	

Method	What it does	Parameter Type	Returns
rotateBoundary	Rotates the boundary of selection	angle as Double *anchor as AnchorPosition* *AnchorPosition.BOTTOMCENTER* *.AnchorPosition.BOTTOMLEFT* *AnchorPosition.BOTTOMRIGHT* *AnchorPosition.MIDDLECENTER* *AnchorPosition.MIDDLELEFT* *AnchorPosition.MIDDLERIGHT* *AnchorPosition.TOPCENTER* *AnchorPosition.TOPLEFT* *AnchorPosition.TOPRIGHT*	
select		region as Object *type as SelectionTypeSelectionType.DIMINISH* *SelectionType.EXTEND* *SelectionType.INTERSECT* *SelectionType.REPLACE* *feather as Double* *antiAlias as Boolean* *antiAlias.CRISP* *antiAlias.NONE* *antiAlias.SHARP* *antiAlias.SMOOTH* *antiAlias.STRONG*	
selectAll			
selectBorder	Selects the border of the selection	width as UnitValue	
similar	Grows selection to include pixels throughout the image falling within the tolerance range	tolerance as Long antiAlias as Boolean *antiAlias.CRISP* *antiAlias.NONE* *antiAlias.SHARP* *antiAlias.SMOOTH* *antiAlias.STRONG*	
smooth		radius as Long	
store	Saves the selection as a channel	into as Channel *combination as SelectionType* *SelectionType.DIMINISH* *SelectionType.EXTEND* *SelectionType.INTERSECT* *SelectionType.REPLACE*	

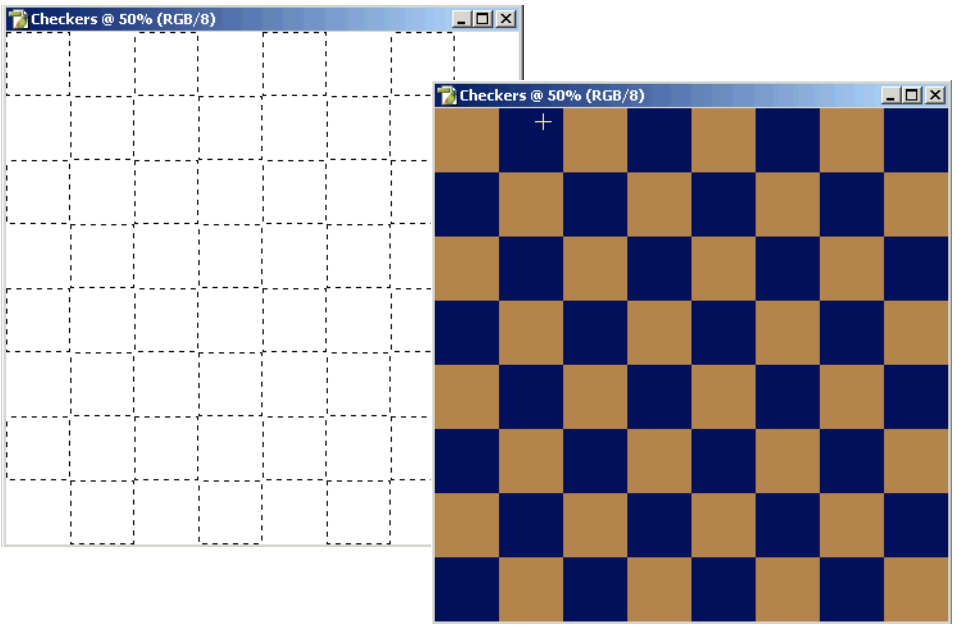
Method	What it does	Parameter Type	Returns
stroke	Strokes the selection	strokeColor as ANYTHING width as Long *location as StrokeLocation* *StrokeLocation.*CENTER *StrokeLocation.*INSIDE *StrokeLocation.*OUTSIDE *mode as ColorBlendMode* *ColorBlendMode.BEHIND* *ColorBlendMode.CLEAR* *ColorBlendMode.COLOR* *ColorBlendMode.COLORBURN* *ColorBlendMode.COLORDODGE* *ColorBlendMode.DARKEN* *ColorBlendMode.DIFFERENCE* *ColorBlendMode.DISSOLVE* *ColorBlendMode.EXCLUSION* *ColorBlendMode.HARDLIGHT* *ColorBlendMode.HUE* *ColorBlendMode.LIGHTEN* *ColorBlendMode.LINEARBURN* *ColorBlendMode.LINEARDODGE* *ColorBlendMode.LINEARLIGHT* *ColorBlendMode.LUMINOSITY* *ColorBlendMode.MULTIPLY* *ColorBlendMode.NORMAL* *ColorBlendMode.OVERLAY* *ColorBlendMode.PINLIGHT* *ColorBlendMode.SATURATION* *ColorBlendMode.SCREEN* *ColorBlendMode.SOFTLIGHT* *ColorBlendMode.VIVIDLIGHT* *opacity as Long* *preserveTransparency as Boolean*	
translate	Moves the position relative to its current position	*deltaX as UnitValue* *deltaY as UnitValue*	
translateBoundary	Moves the boundary of selection relative to its current position	*deltaX as UnitValue* *deltaY as UnitValue*	

Sample Script

The following *selection* script creates a new document by dividing an 800 pixel board into 100 x 100 pixel squares.

The checkerboard is created by iterating through an array of alternating selections in the shape of squares. One alternating selection of squares is filled with a foreground color from the palette. Then the procedure is inverted and the other selection of squares is filled with a background color from the palette. The squares are then de-selected to remove the "marching ants".

The script successively produces the following checkerboards.



Note: For this script to be effective, the foreground and background colors of the current palette much be different colors.

Code (Selection.js)

```
// Save the current preferences
var startRulerUnits = app.preferences.rulerUnits;
var startTypeUnits = app.preferences.typeUnits;
var startDisplayDialogs = app.displayDialogs;

// Set Photoshop to use pixels and display no dialogs
app.preferences.rulerUnits = Units.PIXELS;
```

```
app.preferences.typeUnits = TypeUnits.PIXELS;
app.displayDialogs = DialogModes.NO;

// first close all the open documents
while (app.documents.length) {
   app.activeDocument.close();
}

// 800 pixel board divided in even 100 x 100 squares
var docSize = 800;
var cells = 8;
var cellSize = docSize / cells;

// create a new document
var checkersDoc = app.documents.add(docSize, docSize, 72, "Checkers");

// select the checker board
// every other row I need to shift my selection
// one square to the right, this is done with shiftIt
var shiftIt = true;

// loop through vertically
for (var v = 0; v < docSize; v += cellSize) {

   // i'm on a new row so switch the shift
   shiftIt = !shiftIt;

   // loop through horizontally
   for (var h = 0; h < docSize; h += (cellSize * 2)) {

      // push over the cellSize to start with only
      if (shiftIt && h == 0) {
         h += cellSize;
      }

      // make me a square selection
      selRegion = Array(Array(h, v),
                  Array(h + cellSize, v),
                  Array(h + cellSize, v + cellSize),
                  Array(h, v + cellSize),
                 Array(h, v));

      // if i just started then start the selection
      // otherwise extend the selection
      if (h == 0 && v == 0) {
         checkersDoc.selection.select(selRegion);
      } else {
         checkersDoc.selection.select(selRegion, SelectionType.EXTEND);
      }
```

```
        // turn this off for faster execution
        // turn this on for debugging
        WaitForRedraw();
    }
}

// now I have my selection I will fill with the foreground
checkersDoc.selection.fill( app.foregroundColor );

// invert the selection
checkersDoc.selection.invert();

// and fill with the background
checkersDoc.selection.fill( app.backgroundColor );

// and clear the selection
checkersDoc.selection.deselect();

// Reset the application preferences
app.preferences.rulerUnits = startRulerUnits;
app.preferences.typeUnits = startTypeUnits;
app.displayDialogs = startDisplayDialogs;


// I little helper function I use for debugging
// It also helps the use see what is going on
// if you turn it off for this example you
// just get a flashing cursor for a long time
function WaitForRedraw()
{
    // comment or uncomment the next line
    // to slow down or speed up this action
    // return;
    var eventWait = charIDToTypeID( 'Wait' );
    var enumRedrawComplete = charIDToTypeID( 'RdCm' );
    var typeState = charIDToTypeID( 'Stte' );
    var keyState = charIDToTypeID( 'Stte' );

    var desc = new ActionDescriptor();

     desc.putEnumerated( keyState, typeState, enumRedrawComplete );

    executeAction( eventWait, desc, DialogModes.NO );
}
```

SGIRGBSaveOptions

Properties

Property	Access	Value Type	What it is
alphaChannels	R/W	Boolean	save alpha channels
spotColors	R/W	Boolean	save spot colors

SolidColor

Properties

Property	Access	Value Type	What it is
cmyk	R/W	CMYKColor	return a grayscale representation of the color
gray	R/W	GrayColor	return a grayscale representation of the color
hsb	R/W	HSBColor	return a grayscale representation of the color
lab	R/W	LabColor	return a grayscale representation of the color
model	R/W	ColorMode ColorMode.CMYK ColorMode.GRAYSCALE ColorMode.HSB ColorMode.LAB ColorMode.NONE ColorMode.RGB	color model
nearestWebColor	RO	RGBColor	The nearest web color to the current color
rgb	R/W	RGBColor	return an rgb representation of the color

Methods

Method	What it does	Parameter Type	Returns
isEqual	Returns true if the provided color is visually equal to this color	color as SolidColor	Boolean

SubPathInfo

Properties

Property	Access	Value Type	What it is
closed	R/W	Boolean	is this path closed?
entireSubPath	R/W	Object	all the sub path item's path points
operation	R/W	ShapeOperation	sub path operation on other sub paths

SubPathItem

Properties

Property	Access	Value Type	What it is
closed	RO	Boolean	is this path closed?
operation	RO	ShapeOperation	sub path operation on other sub paths
parent	RO	Object	the object's container
pathPoints	RO	PathPoints	

SubPathItems

Properties

Property	Access	Value Type	What it is
length	RO	Long	number of elements in the collection
parent	RO	Object	the object's container

TargaSaveOptions

Properties

Property	Access	Value Type	What it is
alphaChannels	R/W	Boolean	save alpha channels
resolution	R/W	TargaBitsPerPixels TargaBitsPerPixels.SIXTEEN TargaBitsPerPixels.THIRTYTWO TargaBitsPerPixels.TWENTYFOUR	number of bits per pixel (default: TargaBitsPerPixels.TWENTYFOUR)
rleCompression	R/W	Boolean	should RLE compression be used? (default: true)

TextFont

Properties

Property	Access	Value Type	What it is
family	RO	String	the family of the font
name	RO	String	the name of the font
parent	RO	Object	the object's container
postScriptName	RO	String	this is the string used to assign a font to a text item.
style	RO	String	the style of the font

TextFonts

Properties

Property	Access	Value Type	What it is
length	RO	Long	number of elements in the collection
parent	RO	Object	the object's container

Methods

Method	What it does	Parameter Type	Returns
getByName	Get the first element in the collection with the provided name	name as String	TextFont

TextItem

Properties

Property	Access	Value Type	What it is
alternateLigatures	R/W	Boolean	use alternate ligatures?
antiAliasMethod	R/W	Anti Alias	
autoKerning	R/W	AutoKernType AutoKernType.MANUAL AutoKernType.METRICS AutoKernType.OPTICAL	options for auto kerning
autoLeadingAmount	R/W	Double	percentage to use for auto leading
baselineShift	R/W	UnitValue	baseline offset of text (unit value)
capitalization	R/W	TextCase TextCase.ALLCAPS TextCase.NORMAL TextCase.SMALLCAPS	the case of the text
color	R/W	SolidColor	color of text
contents	R/W	String	the text in the layer
desiredGlyphScaling	R/W	Double	
desiredLetterScaling	R/W	Double	
desiredWordScaling	R/W	Double	
direction	R/W	Direction Direction.HORIZONTAL Direction.VERTICAL	text orientation
fauxBold	R/W	Boolean	use faux bold?
fauxItalic	R/W	Boolean	use faux italic?
firstLineIndent	R/W	UnitValue	(unit value)
font	R/W	String	text face of the character
hangingPuntuation	R/W	Boolean	use Roman Hanging Punctuation?
height	R/W	UnitValue	the height of paragraph text (unit value)
horizontalScale	R/W	Long	horizontal scaling of characters (in percent)

Property	Access	Value Type	What it is
hyphenateAfterFirst	R/W	Long	hyphenate after this many letters
hyphenateBeforeLast	R/W	Long	hyphenate before this many letters
hyphenateCapitalWords	R/W	Boolean	wheter to hyphenate capitalized words
hyphenateWordsLongerThan	R/W	Long	hyphenate words that have more than this number of letters (minimum 0)
hyphenation	R/W	Boolean	use hyphenation?
hyphenationZone	R/W	UnitValue	the hyphenation zone (unit value)
hyphenLimit	R/W	Long	maximum number of consecutive hyphens
justification	R/W	Justification Justification.CENTER Justification.CENTERJUSTIFIED Justification.FULLYJUSTIFIED Justification.LEFT Justification.LEFTJUSTIFIED Justification.RIGHT Justification.RIGHTJUSTIFIED	paragraph justification
kind	R/W	TextType TextType.PARAGRAPHTEXT TextType.POINTTEXT	the type of the text
language	R/W	Language Language.BRAZILLIANPORTUGUESE Language.CANADIANFRENCH Language.DANISH Language.DUTCH Language.ENGLISHUK Language.ENGLISHUSA Language.FINNISH Language.FRENCH Language.GERMAN Language.ITALIAN Language.NORWEGIAN Language.NYNORSKNORWEGIAN Language.OLDGERMAN Language.PORTUGUESE Language.SPANISH Language.SWEDISH Language.SWISSGERMAN	

Property	Access	Value Type	What it is
leading	R/W	UnitValue	leading (unit value)
leftIndent	R/W	UnitValue	(unit value)
ligatures	R/W	Boolean	use ligatures?
maximumGlyphScaling	R/W	Double	
maximumLetterScaling	R/W	Double	
maximumWordScaling	R/W	Double	
minimumGlyphScaling	R/W	Double	
minimumLetterScaling	R/W	Double	
minimumWordScaling	R/W	Double	
noBreak	R/W	Boolean	
oldStyle	R/W	Boolean	use old style?
parent	RO	Object	the object's container
position	R/W	Array(UnitValue)	position of origin (unit value)
rightIndent	R/W	UnitValue	(unit value)
size	R/W	Double	font size in points
spaceAfter	R/W	UnitValue	(unit value)
spaceBefore	R/W	UnitValue	(unit value)
strikeThru	R/W	StrikeThruType StrikeThruType.STRIKEBOX StrikeThruType.STRIKEHEIGHT StrikeThruType.STRIKEOFF	options for strik thru of the text
textComposer	R/W	TextComposer TextComposer.ADOBEEVERYLINE TextComposer.ADOBESINGLELINE	type of text composing engine to use
tracking	R/W	Double	controls uniform spacing between multiple characters
underline	R/W	UnderlineType UnderlineType.UNDERLINELEFT UnderlineType.UNDERLINEOFF UnderlineType.UNDERLINERIGHT	options for underlining of the text
useAutoLeading	R/W	Boolean	whether to use a font's built-in leading information
verticalScale	R/W	Long	vertical scaling of characters (in percent)
warpBend	R/W	Double	percentage from -100 to 100

Property	Access	Value Type	What it is
warpDirection	R/W	Direction Direction.HORIZONTAL Direction.VERTICAL	
warpHorizontalDistortion	R/W	Double	percentage from -100 to 100
warpStyle	R/W	WarpStyle WarpStyle.ARC WarpStyle.ARCH WarpStyle.ARCLOWER WarpStyle.ARCUPPER WarpStyle.BULGE WarpStyle.FISH WarpStyle.FISHEYE WarpStyle.FLAG WarpStyle.INFLATE WarpStyle.NONE WarpStyle.RISE WarpStyle.SHELLLOWER WarpStyle.SHELLUPPER WarpStyle.SQUEEZE WarpStyle.TWIST WarpStyle.WAVE	
warpVerticalDistortion	R/W	Double	percentage from -100 to 100
width	R/W	UnitValue	the width of paragraph text (unit value)

Methods

Method	What it does	Parameter Type	Returns
convertToShape	Converts the text item and its containing layer to a fill layer with the text changed to a clipping path		
createPath	Creates a work path based on the text object		

TiffSaveOptions

Properties

Property	Access	Value Type	What it is
alphaChannels	R/W	Boolean	save alpha channels
annotations	R/W	Boolean	save annotations
byteOrder	R/W	ByteOrder ByteOrder.IBM ByteOrder.MACOS	Default value is 'Mac OS' when running on MacOS, and 'IBM PC' when running on a PC
embedColorProfile	R/W	Boolean	embed color profile in document
imageCompression	R/W	TIFFEncoding TIFFEncoding.JPEG TIFFEncoding.NONE TIFFEncoding.TIFFLZW TIFFEncoding.TIFFZIP	compression type (default: TIFFEncoding.NONE)
jpegQuality	R/W	Long	quality of produced image. Only valid for JPEG compressed TIFF documents (0 - 12)
layerCompression	R/W	LayerCompression LayerCompression.RLE LayerCompression.ZIP	should only be used when you are saving layers
layers	R/W	Boolean	save layers
saveImagePyramid	R/W	Boolean	(default: false)
spotColors	R/W	Boolean	save spot colors
transparency	R/W	Boolean	

xmpMetadata

Properties

Property	Access	Value Type	What it is
parent	RO	Object	the object's container
rawData	R/W	String	raw XML form of file information

7

JavaScript Syntax

JavaScript is a powerful, object-oriented scripting language that was first developed by Netscape Communications to enhance web-page interactivity. Originally named LiveScript, JavaScript actually has very little to do with Java. Although it uses a language syntax similar to Java (or to C, for that matter) it is a language of its own, with rules that are often very different from those found in Java.

JavaScript is an interpreted language. Before you can run your programs in C, C++ or Java, you need to create a source file, then run a compiler program that translates the source file into an executable file containing machine code instructions. In a JavaScript environment, however, all commands and program statements are executed as soon as you type them in.

Originally designed for Netscape's browser software, JavaScript has rapidly evolved to become a widely used, general-purpose programming language. It is now accepted as a standard under ISO-16262 of the International Standards Organization. (The first industry-standard version of the language, endorsed by the European Computer Manufacturers Association, was known as ECMAScript). The core language has undergone several revisions, the most current being version 1.5.

JavaScript is designed to use the Unicode character set. Therefore, you are free to use your local characters as long as they fit into the Unicode character set.

Core JavaScript Language Features

In this section, a brief discussion of some of the syntax rules used in JavaScript is given.

Identifiers

An identifier is a name that appears in JavaScript code. Identifiers are used for the names of variables, functions and labels. An identifier must begin with a letter, an underscore or a dollar sign, subsequent characters can also include digits. Thus, *myVar1*, *_myFunction*, *$my_Var17* are all legal identifiers. Identifiers can contain Unicode characters, so the identifier "" is perfectly legal.

Case Sensitivity

The JavaScript language is case sensitive, the identifier *myField* is considered different from *myfield*. As a result, great care must be taken when typing program statements, an awareness of the case sensitivity of the language is very important.

Semicolons

The semicolon (;) is used to separate JavaScript statements. If the statements are on separate lines, the semicolon is optional.

```
x = 1
y = 2
```

If the statements are on the same line, which is not good practice, the semicolon separator is required:

```
x = 1; y = 2
```

Using a semicolon at the end of each statement is good practice:

```
x = 17;
y = x + 1;
z = y * y;
```

Comments

Comments can be inserted into JavaScript code using either the C++ or C-style commenting protocol. Any text between a double-slash (//) and the end of a line will be ignored by JavaScript. Also, any text between /* and */ will be treated as a non-executable item (or comment). The following are valid comment styles:

```
// This is a single-line comment.
x = 1;
```

```
/* This is also a comment. */
y = 2; // as is this
/*
*
* An extended comment
*
*/
z = 3;
```

Comments are extremely important tools because they make programs much more readable and easier to maintain. JavaScript uses a succinct, C-like syntax, which means a lot can be accomplished in just a few lines of code.

JavaScript can be quite compact, but nearly indecipherable and hard to debug. Comment the code liberally as it is written: explain the goal of each code segment, document the parameters used, list any dependencies, or explain the reasoning used to write the code. The resulting commented code will make it much easier to read, maintain, extend or debug at a later time.

Data Types

JavaScript supports primitive (core) data types—such as booleans, numbers, and strings—as well as composite data types, like objects, arrays and functions. The primitive data types are used most often; consequently, it is important to understand these types and how JavaScript interprets them.

Primitive Data Types

Booleans are the simplest data type, since they can have just two values, *true* and *false*. Internally, JavaScript represents true and false values as 1 and 0, respectively. But anything that has a non-zero value will be evaluated by the JavaScript interpreter as true in a Boolean context.

Numbers are represented using standard scientific notation, for example, 17, -88, 3.14159, or 6.023e+23. Unlike some other languages, JavaScript does not distinguish between integer or floating-point values. In JavaScript, all numbers are represented internally as 64-bit IEEE floating-point values. In base-ten terms, there are about 20 digits of precision in which to work, which ought to be adequate for most applications.

Note that integer values should not be written with leading zeros in JavaScript. JavaScript interprets '021' as the octal representation of the base-ten number 17. (In addition to octal triplets, hexadecimal numbers can be represented using the '0x' prefix plus two hexadecimal digits; for example, 0xFF represents a value of 255.)

A *string* is a sequence of letters, digits, punctuation, or other characters enclosed in quotation marks. Single quotations or double quotations can be used, it doesn't matter as long as they match. (If a string contains double-quotes as part of the desired string sequence, enclose the entire string in *single* quotes. Likewise, if a string contains single-quotes as part of the string sequence, enclose the entire string in *double* quotes.) The following are all legal string values:

```
'This will work.'
"3.14"
'The password is "xyzzy"'
```

Strings can be concatenated using the "+" operator. For example

```
"Roses are red " + "and Violets are blue."
```

JavaScript recognizes a number of *escape sequences* for representing characters inside strings that would otherwise be impossible to represent. The following table summarizes those escape sequences.

Escape Sequence	Description
\b	Backspace
\f	Form feed

Escape Sequence	Description
`\n`	Newline
`\r`	Carriage return
`\t`	Tab
`\`	Single quote
`\`	Double quote
`\\`	Backslash
`\xXX`	Character specified by two hexadecimal digits
`\uXXXX`	Unicode character specified by four hexadecimal digits

Escape sequences can be exploited to better handle strings with embedded single or double quotation marks in them, for example:

```
"\"Quoth the raven, \'Nevermore!\'\""
```

Composite data types

JavaScript defines another data type called an *object*, a structure used for holding a collection of name/value pairs. The values held in the object can be accessed through its (associated) name. The names are referred to as the *properties* of the object. The value of a property can be a primitive data type, or another object.

Other composite data types, such as *arrays* and *functions*, are special types of objects. Though arrays and functions are objects, JavaScript defines a special syntax for handling each of them. These composite data types are discussed in the following sections.

Declaring and Accessing Variables

Some important points to keep in mind concerning JavaScript variables are listed below:

- All variable names in JavaScript are case-sensitive, which means that a variable named *MyVariable* is not the same as one named *myVariable*.
- A variable name, which is an identifier, must begin with a letter, an underscore, or a dollar sign. Subsequent characters in a variable name can be letters, digits, underscores or dollar signs. Thus, *address1*, *address2*, *_name*, *$income* are all valid variable names. Since JavaScript supports Unicode, the usage of Unicode characters in a variable name is OK as well.
- Variables created in JavaScript are permanent, within their scope. Once a variable is declared, you can use the *delete* operator to delete a variable. For variables declared with the *var* statement, there is no way to manually "undeclare" or destroy them. (JavaScript's

garbage-collection mechanism will automatically de-allocate variables when they are no longer needed.)

- All variables in JavaScript have a scope which determines the variable's lifetime and accessibility. A variable declared inside a function is said to have *local scope*, which means it can be used only inside the function in which it was declared.

Declaration of variables can be made by typing the keyword '*var*' followed by the name of the variable being declared. For example,

```
var radius;
```

Several declarations can be made by separating each variable declaration by a comma. For example,

```
var radius, pi, circumference, area; // only one var needed here
```

Declaring a variable, and initializing its value can be accomplished by using the “assignment” operator. For example,

```
var radius = 2, pi = 3.14, circumference, area;
```

The following example declares variables for different primitive data types:

```
var creditCard = false; // Boolean
var cost = 19.95; // number
var name = "John Doe"; // string
```

After declaring a variable and assigning it a value, it is no longer necessary to preface it with the *var* keyword. The data is simply accessed through the variable name.

```
var radius = 2, pi = 3.14, circumference, area;
area = pi * radius * radius;
circumference = 2 * pi * radius;
var strArea = "The area of the circle is " + area;
```

Here, a string is concatenated with a number. In this case, the JavaScript interpreter converts *area*, a number type, into a string type.

It is legal, in JavaScript, to declare a variable without using the 'var' keyword. But since (as we mentioned) all JavaScript variables must have a scope, this leaves the interpreter in a bit of a quandary as to how to "scope" a non-'var' variable. The interpreter resolves this problem by attaching the unscoped variable to the global name space, which has the effect of making the variable in question "usable" from all points in a program.

Undefined Variables

In JavaScript, as in other languages, any attempt to use a variable in an expression without first declaring it generates an exception. Consider the following code:

```
var x = 1;
z = x + y;
```

where y is a new variable that was not declared anywhere. When these two lines are executed, a JavaScript exception *ReferenceError* would be thrown.

JavaScript has relaxed typing rules, but this is not the same as saying that it is an untyped language. JavaScript does have data types. The operator *typeof* can be used to identify the data type of a variable. To test if the variable y has been defined in the document, execute the following script:

```
if (typeof y == "undefined")
    alert("Undefined variable.");
else {
    // the variable is safe to use
}
```

Here, the predefined alert method is used to put an alert dialog onscreen if the variable being tested is not defined. Notice that JavaScript uses the C-like "== operator to test for equality. Also note that strings are compared *by value*; hence two strings can be compared directly using the "== operator.

Variables declared with the *var* keyword do not follow this rule, because the JavaScript interpreter creates these variables as the var statement is executed. If the var statement did not assign a value to the variable, its contents are undefined, which is not equal to the variable being undefined.

Actually, JavaScript treats undefined data as being a special value. The predefined global variable *undefined* contains this value, and the result of non-existing object properties (as you will see below) is reported as *undefined*.

Operators

As in many other programming languages, a large number of operators are built in to JavaScript. The following table details the available operators; you may notice that there are a few operators very specific to the JavaScript language.

Unary operators

These are operators that apply to one operand only.

Unary Operators	Description
`delete`	Delete a variable. The *delete* operator applies to a variable or property reference, like e.g. *hoopla*, *yArray[3]* or *myObject.prop*. Note that the *delete* operator cannot delete variables declared with the *var* statement. If the variable contains an object, the object is not destroyed immediately; the garbage collector destroys it once all references to the object are gone. The value of the *delete* operation is either *true* or *false*, depending on whether the operand could be deleted.
`void`	This operand instructs JavaScript to forget about the results of an operation. Usually, an operation like 4+5 would be 9, but *void(4+5)* is Undefined.
`typeof`	Return the type of its operand. The result is one of the strings "undefined", "Boolean", "number", "string", "function" or "object".
`+, -`	Unary plus and minus.
`~`	Bitwise inversion.
`!`	Boolean negation.
`++, --`	Increment and decrement. This operator is either a prefix operator (++i) or a postfix operator (i++). The result depends on the type of operator. For a prefix operator, the operand is first incremented (or decremented), and the result is the outcome of that operation. For a postfix operation, the result is the contents of the operand before the operation has been applied.

Binary operators

Binary operators concatenate two operands. They are sorted in their order of precedence. If you are not sure whether one operator takes precedence of the other, use brackets. This example demonstrates the usefulness of brackets:

```
a instanceof String ? b / c : typeof d
```

This has the same meaning, but is far more readable:

```
(a instanceof String) ? (b / c) : (typeof d)
```

Binary Operators	Description
`* / %`	Multiplication, division and modulus.
`+ -`	Addition and subtraction. The '+' operator applied to a string converts its arguments to strings and concatenates them.
`<< >> >>>`	Bitwise shift operators. The left-hand side operand is converted to an integer and shifted left or right as indicated by the right-hand operand. The right-shift operation is either signed (>>) or unsigned (>>>).
`< > <= >=`	Comparison operators. The comparison depends on the type of operators. If both are strings, the comparison is a string comparison, otherwise, JavaScript attempts to convert both operands to a number and to perform a numeric comparison.
`instanceof`	Test whether the class of the object given as the left-hand operator is an instance of the class given as the right-hand operator. The right-hand operator must be the name of a global class constructor function, like e.g. *String* or *Object*. The result is a Boolean value.
`in`	Test whether the left-hand operator is a property of the right-hand operator. The left-hand operator must either evaluate to a string or a number, and the right-hand operator must be an object. The result is a Boolean value. All of the comparison operators form a group of the same operator precedence weight.
`== !=`	Equality operators. JavaScript attempts to convert both operand to the same type before comparing them.
`=== !==`	Identity operators. JavaScript does not attempt to convert both operands to the same type before comparing them. The identity and equality operators have the same precedence weight, however.
`& | ^`	Bitwise AND, OR and XOR. Both operands are converted to integers before applying the operation, and the result is a number containing the resulting bits.
`&& ||`	Boolean AND and OR. Both operands are converted to Boolean operands before applying the operator. If the left-hand operand already satisfies the condition (like being *false* on an AND or *true* on an OR operation), the right-hand side operator is not evaluated. The result is the Boolean combination of the operands.

Ternary operators

The ternary operator is actually an abbreviation of the *if* statement. It has the syntax:

```
condition ? true-expression : false-expression
```

The condition is evaluated, and, it if it evaluates to *true*, the result of the operation is the result of the *true-expression*; otherwise, the result is the result of the *false-expression*. An example:

```
a = i > 5 ? "Yes" : "No";
```

The variable *a* contains "Yes" if *i* is greater than 5, or "No" otherwise.

Assignment operators

The assignment operators are almost at the very bottom of the operator precedence list. Assignment operations have a value, which is the value of the right-hand operand. Therefore, a statement like "a=b=c=0" makes perfect sense.

JavaScript has the simple assignment operator "=" as well as compound assignment operators, which are the assignment operator combined with an arithmetic or logic operator:

```
*= /= %= += -= <<= >>= >>>= &= |= ^=
```

These operators are a short form of

```
left-hand = left-hand op right-hand
```

Defining and Using Objects

The treatment of objects will be highly abbreviated. For a more in depth discussion, refer to any good book covering core JavaScript.

A convenient way of defining an object is the *object literal*. An object literal is a collection of name/value pairs, set apart by commas, and enclosed in matching curly braces. The name/value pairs are separated by a colon. For example:

```
{ question: "How are you today?", title: "Your Health Status"}
```

This defines an object with two *properties*; the property names are *question* and *title*. The order in which the properties are listed is immaterial. Objects can be assigned to variable names, just as primitive data types can:

```
var params = { question: "How are you today?",
           title: "Your Health Status" };
```

Data within the object can be accessed through the "." notation. As opposed to simple variables, variables as part of an object are called *properties*. For example, the value of the question property of the params object *is params.question*. It is important to note *that params.question*, in this example, behaves much like a variable. Here, *params.question* represents the string `"How are you today?"`.

Values of existent properties can be modified easily:

```
params.question = "How were you yesterday?";
```

Properties can be added to the object as well:

```
params.dflt ="Fine";
```

Note that you can also use numeric or string constants as property names to define an object; it depends on whether the property name conforms to the syntax rules of identifiers or not. An example:

```
var myObj = { 5: "Five", "John Doe": "My name" };
```

The [] operator can also be used to access the properties of an object. For example, *params["question"]* represents the string "How are you today?". Note that the expression enclosed in the brackets is a string, not an identifier. As will be seen in the paragraphs on the for/in Loop, this method of accessing the properties and values of an object is quite useful, and, if the above example, it is the only way to access the properties of *myObj*.

As was mentioned before, the value of a property can be a primitive data type, a *function*, an array, or another object. If a property of an object has as its value a function, that property is referred to as a *method of the object*. JavaScript defines a number of objects, each having properties and methods.

Example: JavaScript defines the *String* object. *String* object have a number of methods that work on the string contained in the *String* object, like e.g. the *charAt* method The method is accessed in the same way as a property, using the "." notation, for example,

```
s = new String ("Hi world");
ch = s.charAt (0);// ch now contains the string "H"
```

You can also define objects the procedural way, using the *Object* constructor function:

```
var params = new Object;
params.question = "How are you today?";
params.title = "Your Health Status";
```

Declaring and Using Arrays

An object, as described in the previous paragraphs, is an *unordered* collection of data. An array, is an *ordered* collection of data. The data is indexed by the nonnegative integers. There are two

methods of declaring an array that will be discussed in these paragraphs. Arrays can be declared using an *array literal*, or an *array constructor*.

Example:

```
// array literal
var myArray1 = [ "This", "is", "ExtendScript", 3.0 ];
var myArray2 = [ 1, "String", { x:1, y:2 } ];
```

Note that the *myArray2* contains a mixture of data types: number, string and object.

```
// Array constructor
var colors = new Array("red", "green", "blue");
```

The indexing system is 0-based, the first element of the array has index 0, the second element has index 1, and so on. To access the elements of an array, the [] operator is used, with the index number inserted between the brackets. For example

```
var str = myArray1[0]; // now str = "This"
var version = myArray1[3]; // version = 3.0
// The element myArray2[2] is an object, to access it,
// the dot notation is used to access property y
var z = myArray2[2].y; // z is initialized to a value of 2.
```

The declaration *var myArray = new Array()* creates an empty array of length zero. The elements of *myArray* are undefined:

```
if (typeof myArray[0] == "undefined")
    alert("No such element"); // this message appears
else
    alert("There is a 0th element in the array");
```

One the empty array is declared, elements can be added to it; for example, after the code lines

```
myArray[0] = "Adobe";
myArray[1] = "Acrobat";
```

are executed, *myArray* is an array of length two.

There is no need to supply each element of the array. If you use two consecutive commas, the element that is missing remains undefined. Note, however, that you will have to use two commas at the end of the array literal if you want the last array element to be undefined. Examples:

```
myArray = ["One", "Two",,"Four"];// myArray[2] is undefined
myArray = ["One", "Two", "Three", ];// a three-element array
myArray = ["One", "Two", "Three",, ];// a four-element array
```

Regular expressions

JavaScript contains a full regular expression object which you can use to parse strings in very complex ways. The full syntax of regular expressions would be too much for this document; please refer to more elaborate books about JavaScript. There is a way to declare a regular expression by enclosing it in slashes:

```
var regex = /a*/i
```

This statement creates a *RegExp* object. The object has a number of methods and properties, the *exec()* method being the most important one. That method returns an array of matches, the first element being the match found, and other elements containing the result of enclosed regular expression captures. Examples:

```
var regex = /a+/i
regex.exec ("Bart Simpson"); // returns ["a"]
/(\w*) (\w*)/.exec ("Bart Simpson");
// returns ["Bart Simpson", "Bart", "Simpson"]
```

Functions

A *function* is a block of JavaScript code that is compiled once, but can be executed many times. Some of the important points concerning functions include:

- Parameters can be passed to a function
- The function may have a return value
- Within the body of the function definition, any variables declared with the *var* keyword have local scope.

There are a number of ways of defining a JavaScript function. Only the most popular style is presented here. The syntax for defining a function is

```
function functionName (parameter list)
{
JavaScript statements
}
```

The labels *functionName* and *parameter list* are replaced by the name of the function that is being defined, and by the list of parameters of the function, respectively. The following defines a function *mySum* and will be used to illustrate the itemized points above.

```
function mySum (x, y) // parameters x and y
{
var z = x + y; // add the two parameters
return z; // use the sum as the return value
}
```

Once this function is defined, it can be called, consider the following:

```
var sum = mySum(10, 7); // sum = 17
```

Here, the values 10 and 7 are passed to the function *mySum* as its required parameters. The function then returns the sum of the two passed parameters. In the body definition, a *local variable* z is declared. The (local) variable z is unknown outside this function. (Recall the assumption of this example that the variable z has not been declared anywhere at the top level outside the function.) To demonstrate this last statement, executing the line

```
var str = "The value of z is " + z;
```

results in a *ReferenceError* being thrown.

A function need not have any parameters or a return value. For example:

```
function helloWorld() {
alert("Hello world");
}
```

The *helloWorld* function simply pop ups an alert box on the screen.

If a function is attached to an object, it is called a *method*. You can attach a function to an object by assigning the function name to the property or by declaring the function directly:

```
var obj = new Object();
obj.foo = helloWorld;// note the missing brackets ()
obj.bar = function() {
          alert ("This is a wonderful world!");
         }
```

The call *obj.foo()* would pop up the "Hello world" alert box, and the call *obj.bar()* would pop up the alert "This is a wonderful world!".

Actually, a function is just a *Function* object; therefore, you can attach it to whatever object you like, or move it around, or even store it into an array. Consider this example of an array literal:

```
fooArray = [
    function() { return "One"; },
    function() { return "Two"; },
    function() { return "Three"; }
          ];
```

This creates an array containing three functions. You could call any of these functions by indexing the array as usual:

```
fooArray [0]();// would return "One"
```

If a function is attached to an object, it can use the *this* keyword to access the object that it is attached to. An example:

```
obj = { value: 5, getValue: function() { return this.value; } };
```

The call *obj.getValue()* would return 5. If the *this* keyword was omitted, it would attempt to locate the variable *value* in its scope chain; if it was executed within a function body containing the variable *value* defined as *var*, it would use that value, or it would use the value of the global variable *value* if present; so the results may become pretty unpredictable.

Predefined variables and functions

The global namespace of JavaScript contains a few variables and functions that are worth mentioning.

Variables and Functions	Description
undefined	This variable always contains the value *undefined*.
NaN	This variable contains the special numeric value Not-A-Number. The value is used for undefined arithmetic operations, like *Math.sqrt (-1)*.
Infinity	This is the value for +∞.
eval (text)	Evaluate and run the JavaScript program scriptlet. *eval ("a=5")* has the same meaning as *a=5*.
parseInt (text)	Convert the given text to an integer number. Optionally, you can supply a conversion radix. *parseInt ("12", 16)* would return 18.
parseFloat (s)	Convert the given text to a floating point number.
isNaN (n)	Check whether the number is Not-A-Number.
isFinite (n)	Check whether the number is a finite number, which is a number between -∞ and +∞.

this	Yes, this keyword also works with the global name space. The global name space is actually an object containing methods and properties, and the *this* keyword points to the global object. Since undefined properties do not throw an error if accessed, you could use the following code to work with global variables without having to care about a *ReferenceError* being thrown if the variable is undefined:
	`var global = this;`
	`a = global.notThere;// assigns Undefined`

Predefined Core Objects

An object is a collection of named values, called the properties of the object. Properties of an object can be any type: strings, numbers, boolean, arrays, functions and even other objects. If a property of an object is a function, it is referred to as a *method* of that object. In JavaScript, a number of predefined objects are available: *Array, Boolean, Date, Function, Math, Number, RegExp, String*. These objects, in conjunction with their properties and methods, can be used, for example, to manipulate arrays and dates, make advanced math calculations, and search and manipulate strings.

The String object has a number of methods for searching and replacing strings. Some of these methods use regular expressions. The *RegExp* object also has methods for searching strings.

Example: Math Object.

```
str1 = "The area of a circle of radius 1 is " + Math.PI;
str2 = "The sine of 45 degrees is " + Math.sin(45);
```

In addition to these core objects, applications usually define a large number of objects, again, each with their own properties and methods used for manipulating the objects.

Conditionals and Loops

Two important program constructs present in all programming languages are conditional statements and looping.

The Conditional Statement

The syntax for a conditional statement is

```
if (expression)
    statement // to be executed if expression is true
else
    statement // to be executed if expression is false
```

This construct is very important. The parentheses that surround *expression* are required. If multiple statements need to be executed, make sure that matching braces enclose the statements.

Examples:

```
if (i == 1)
    alert("The variable i equals 1");
else
    alert("The variable i has a value of " + i);
// testing two conditions: if str equals "Yes" and i is greater than 4
if ((str == "Yes") && (i > 4))
    alert("You are granted access!");
else // here, str != "Yes" or i <= 4
    alert("You may not enter!");
```

When testing for equality, the == (equality) or === (identity) operator can be used; for testing inequality, use != (inequality), or !== (non-identity). Other comparisons include < (less than), > (greater than), <= (less than or equal) and >= (greater than or equal to).

Compound logic can be accomplished using the logical operators of && (and), || (or) and ! (not).

JavaScript has precise operator precedence; however, in the above example, the logical expressions are grouped using parentheses to be certain the logic is correct.

Another conditional construct, the *switch*, can execute code based on a series of mutually exclusive and exhaustive cases. The syntax for the *switch* statement is illustrated below:

```
// Assume the variable, myNum, has some numerical
// value when the code below executes
var msg;
switch (myNum) {
    case 1:  msg = "Case 1";
       // possibly other statements
```

```
        break; // break out of case study
    case 2:  msg = "Case 2"; break;
    case 3:  msg = "Case 3"; break;
    default: msg = "This is the default case";
}
alert(msg);
```

The *switch* statement evaluates the expression, *myVar*, in this case, and tries to match it up with one of the values listed in the *case* statements. The *switch* statement uses the *identity* operator, ===, to make the comparisons; consequently, if myVar = "1" (a string value), the above switch would evaluate to the default case, since (1 === "1") is *false*. A workaround would be to convert *myVar* to a number type.

You can use multiple *case* statements, like this:

```
switch (myNum) {
    case 1:
    case 2:
    case 3:  msg = "Between 1 and 3"; break;
    default: msg = "This is the default case";
}
alert(msg);
```

Or, you can use arbitrary complex expressions to express your *case*:

```
var one = 1, two = 2;
switch (myNum) {
    case one:      msg = "Case 1"; break;
    case two:      msg = "Case 2"; break;
    case one+two:  msg = "Case 3"; break;
    default:       msg = "This is the default case";
}
alert(msg);
```

Loops

Loops enable a block of code to be executed repeatedly under (possibly) different conditions each time.

The *while* loop

The *while* loop is a very common way of looping. The general form for the loop is:

```
while (condition) {
// loop body (JavaScript statements)
}
```

The loop runs repeatedly until the *condition* is false. JavaScript tests the condition before it executes the statements inside the loops. An alternate version tests the condition after the statements inside the loop have been run.

```
do {
// loop body (JavaScript statements)
} while (condition);
```

You can leave a loop with the *break* statement, and you can skip over the remaining statements inside the loop and have the next check processed with the *continue* statement.

The *for* Loop

The *for* loop is a very common way of looping. The general form for the *for* loop is

```
for (initialize; test; increment) {
// loop body (JavaScript statements)
}
```

The *initialize* part of the for loop is used to initialize some variables that are used in the loop. (Multiple initializations must be separated with a comma). The *test* component is evaluated in a boolean context: If *test* evaluates to *true*, the loop body is executed, otherwise, the loop is terminated. After the loop body has been executed, the *increment* expression is evaluated. The *increment* expression is usually some sort of assignment, for example, i=i+1. (An assignment of the form i=i+1 is quite common; the more compact ++ operator is often used. The assignment i=i+1 is equivalent to i++). The enclosing parentheses are required; the grouping braces are required only if there are multiple lines within the body of the loop. Prematurely exiting a loop can be accomplished by executing the *break* statement, and you can use the *continue* statement to skip the remainder of the statements inside the loop and proceed to the next increment operation.

Example: Search through an array for a particular element.

```
var colors = [ "red", "green", "blue" ];
var msg = "I'm thinking of three colors, guess one of them";
var response = prompt (msg, "");
if (response != null) {
    response.toLowerCase();
    for (var i = 0; i < colors.length; i++) {
        if (colors[i] == response)
            break;
       }
        if (i < colors.length)
            alert("You found one!");
          else
            alert("Guess again");
}
else
    alert ("Why don't you guess?");
```

Comments on the Example: After the initial three declarations, the *prompt* method is used to acquire a response from the user. The return value of this method is assigned to the variable *response*. The documentation for this method states that the return value will be the null value if the user cancels the dialog. A conditional statement is used, the search will occur only if *response* is non-null.

The three components of a for loop appear in the above example:

- **Initialize:** The variable i is used to index the loop and it is initialized to 0 with the *statement var i = 0*.
- **Test:** The testing condition is *i < colors.length*. The loop body is repeatedly executed until the index *i* is equal to the length of the colors array.
- **Increment:** After the loop body is executed, the increment expression, *i++*, is evaluated, which increases the value of the index *i* by one. At this point, the test is evaluated and the body of the loop is executed again, *if i < colors.length*, or the loop is exited, if *i >= colors.length*.

In the loop body, note the use of the *break* statement. If the i^{th} array element favorably compares with the user's response, *break* is executed. If the user did not guess one of the colors, then on exit from the loop, the value of *i* will be *colors.length*. This fact is exploited to determine whether the user successfully guessed one of the colors.

The *for/in* Loop

The *for/in* statement can be used to loop through an object. The syntax for this loop is:

```
for (variable in object) {
JavaScript statements
}
```

The enclosing braces are only required it there are multiple lines of JavaScript code contained within the loop body. Again, you can use the *break* statement to exit the loop, or the *continue* statement to skip the remaining statements inside the loop.

The for/in loop creates an internal snapshot of the object's properties. Every time the loop is run, the loop variable receives the name of the next property of this snapshot. You can then use the [] notation to access the contents of that property. Note, however, that most properties and method of built-in object types are not enumerable; you cannot reach them in a for/in loop.

For example, consider this declaration:

```
params = { question: "How are you today?",
    title: "Your Health Status",
    dflt: "Fine" };
```

Executing the following code,

```
for (var p in params)
alert("params." + p + " = " + params[p]);
```

yields the following results:

```
params.question = How are you today?
params.title = Your Health Status
params.dflt = Fine
```

Since an array is also an object, arrays can be searched using the *for/in* loop as well.

Making code readable: the *with* statement

The *with* statement makes code more readable by specifying the object that the statement operates upon so you do not have to use the *object.property* notation. Imagine an array of objects that you need to work on, each containing the same property:

```
var data = [
    { value: 111 },
    { value: 222 },
    { value: 333 }
          ];
function sum (objArray) {
    var result = 0;
    for (var i = 0; i < objArray.length; i++) {
       with (objArray [i]) {
          // instead of objArray[i].value
          result += value;
       }
    }
    return result;
}
alert (sum (data));
```

Dealing With Exceptions

JavaScript methods throw an exception if a run-time error occurs. You can better control these exceptions by using *try, catch* and *finally* statement blocks, possibly in conjunction with the *throw* statement.

The following code defines a method that attempts to access the variable *notThere*, causing a *ReferenceError* to be thrown. The statement below would try to execute the function and catch the thrown error.

```
function testMe() {
    return notThere;
   }
 try {
    testMe();
   }
catch (e) {
    alert (e);
   }
finally {
    alert ("OK, I am done.");
   }
```

The *finally* statement is always executed, either after a successful try or after a successful catch operation.

Error handlers can be written to deal with exceptions thrown by the application and by custom written JavaScript code. A very common case for throwing an exception is if the underlying object no longer exists inside the application (i.e. it is dead, but a reference to the object still exists). Consider the following piece of code:

```
var myDoc = app.newDoc();
myDoc.closeDoc();
myDoc.pageNum = 3;
```

This will throw an exception when the third line is executed. The document has been closed and no longer exists. A reference to this document is still being held in *myDoc* and any attempt to use it will throw an exception.

If you want to catch multiple different errors or other objects, you can use multiple catch clauses. The argument of a catch clause can be expanded with an *if* statement:

```
catch (if condition) {
    statements
   }
```

Note that, unlike the regular *if* statement, the conditional expression is not surrounded by brackets.

For example,:

```
function throwIt() {
    throw "Ouch!";
   }
 try {
    throwIt();
    alert ("Nothing appeared to be thrown");
   }
catch (e if e instanceof Object) {
    // catch all objects
    alert ("An object was caught");
   }
catch (e if e == "Ouch!") {
    // catch the string "Ouch!"
    alert ("Ouch! This hurt!");
   }
catch (e) {
    // catch everything else
    alert ("Caught " + e);
   }
```

As demonstrated, a JavaScript program can throw anything, from simple numbers or strings to complex objects. JavaScript will throw runtime errors as Error objects; there are a number of specialized objects like *ReferenceError* or *TypeError* that are derived from *Error*. Finally, you have to use an unconditional *catch* clause as the last of your *catch* clauses.

Coding conventions

This section contains a set of recommendations for writing JavaScript programs. It is organized as a set of rules with explanations. You are strongly encouraged to follow these rules.

- Use interCaps instead of Underscores

Do not use underscores inside a property or method name to separate parts of the name. Use interCaps instead. Instead of *append_data*, use *appendData*. Which gets us to the next rule:

- Always start a property or a method name with a lowercase letter

In JavaScript, symbols starting with an uppercase letter are considered class names. So, do not use *Document* as a property name, but use *document*, and have that property return a *Document* object. Therefore:

- Always start a class name with a uppercase letter

Class names, like *String* or *Boolean*, should always start with an uppercase letter so you can distinguish class names from property or method names. Analog to class names, there are special constructor functions in the global namespace that carry the same name as the class name, like *String()* for the *String* class etc.

- Do not pollute the global namespace

Define as few global functions and properties as possible, since users will be happy to add their own stuff. This reduces confusion and possible name collisions. Use objects to group names instead. Do not, for example, create global functions or properties that deal with the application. Create an Application object and create a global property *app* instead, containing an *Application* object with all of your application-specific properties.

- Make sure to use *undefined* for missing arguments

Unfortunately, JavaScript does not support missing arguments in the form

```
myDoc.addLayer ("My Layer",,"Initial text");
Do Not Use null or "" (an empty string) to indicate missing arguments!
```

JavaScript defines missing arguments as being undefined. The value null indicates "no object", and the empty string is clearly a string. The variable *undefined* is available for these purposes:

```
myDoc.addLayer ("My Layer",null,"Initial text"); // WRONG!
myDoc.addLayer ("My Layer","","Initial text");   // WRONG!
// THIS IS CORRECT!
myDoc.addLayer ("My Layer",undefined,"Initial text");
```

Remember that *undefined* refers to the contents of a global variable named "undefined". Its value is a special value that JavaScript considers to be an undefined value.

Index