ADOBE® ILLUSTRATOR® CS3

SCRIPTING GUIDE

Adobe®

Adobe® Illustrator® CS3 Scripting Guide for Windows® and Mac OS®.

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA.

Contents

1 Introduction

This guide describes the scripting interface to Adobe® Illustrator® CS3. It contains the following sections:

- This introduction, which describes scripting support in Adobe Illustrator CS3, and lists changes to the scripting interface since the previous release.
- "The Illustrator Scripting Object Model" on page 15, which describes the Illustrator document object model.
- "Scripting Illustrator" on page 23, which provides an overview of how to use scripts to program Adobe Illustrator CS3.
- "Scripting with JavaScript" on page 31, which provides information about scripting Illustrator using JavaScript.
- "Scripting with AppleScript" on page 40, which provides information about scripting Illustrator using AppleScript.
- "Scripting with VBScript" on page 47, which provides information about scripting Illustrator using VBScript.

If you are new to scripting or would like basic information about scripting and how to use the different scripting languages, see *Adobe Introduction to Scripting*.

What is scripting?

A script is a series of commands that tells Illustrator to perform one or more tasks. These tasks can be simple, and affect only a single object in the current document; or complex, and affect objects in all of your Illustrator documents. The tasks might even involve other applications, such as word processors, spreadsheets, and database management programs.

The building blocks of scripting correspond for the most part to the Illustrator tools, menus, palettes, and dialog boxes with which you are already an expert. If you know what you'd like Illustrator to do, you can write a script to do it.

Why use scripting?

Graphic design is a field characterized by creativity, but aspects of the actual work are anything but creative. In fact, you'll probably notice that the time you spend placing and replacing images, correcting errors in text, and preparing files for printing at an image setting service provider often reduce the time you have available for doing creative work.

With a small investment of time and effort—perhaps no more than you'd spend training an assistant—you can learn to write short, simple scripts that perform repetitive tasks for you. As your scripting skills grow, you can move on to more complex scripts that work all night while you're sleeping.

Scripting can also enhance your creativity by quickly performing tasks you might not have time to try. For example, you could write a script to systematically create a series of objects, modifying the new objects' position, stroke, and fill properties along the way. You could also write a script that accesses built-in transformation matrix functions to stretch, scale and distort a series of objects. Without scripting, you'll likely miss out on the creative potential of such labor-intensive techniques.

What about actions?

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

- Actions use a program's user interface to do their work. As an action runs, menu choices are executed, objects are selected, and recorded paths are created. Scripts do not use a program's user interface to perform tasks, and can execute faster than actions.
- Actions have very limited facilities for getting and responding to information. You cannot add conditional logic to an action. Therefore, actions cannot make decisions based on the current situation, such as changing the stroke type of rectangles but not ellipses. Scripts are capable of getting information and making decisions and calculations based on the information they receive from Illustrator.
- A script can execute an action, but actions cannot execute scripts.

Script support in Adobe Illustrator CS3

Illustrator scripting supports AppleScript and JavaScript scripts for Mac OS, or VBScript and JavaScript scripts for Windows.

Note: Additionally, Adobe scripting-enabled applications, including Illustrator, support ExtendScript, Adobe's extended implementation of ECMA JavaScript. ExtendScript files are distinguished by the `.jsx` extension. Giving your JavaScript files a `.jsx` extension allows you to take advantage of the ExtendScript features and tools.

ExtendScript features

- ExtendScript offers all standard JavaScript features, plus a development and debugging environment, the ExtendScript Toolkit (ESTK). The ESTK is installed with all scriptable Adobe applications.
- The ESTK includes an Object Model Viewer that contains complete documentation of the methods and properties of JavaScript objects.

Note: For information on accessing the ESTK and the Model Viewer, see "Viewing the JavaScript object model" on page 8.

ExtendScript tools

ExtendScript also provides various tools and utilities such as:

- A localization utility
- Tools that allow you to combine scripts and direct them to particular applications
- Platform-independent file and folder representation
- Tools for building user interfaces to your scripts
- A messaging framework that allows you to send and receive scripts and data among scripting-enabled Adobe applications

For details of these and additional features, see the *JavaScript Tools Guide*.

Script file extensions

For a file to be recognized by Adobe Illustrator CS3 as a valid script file, the file must have the correct file name extension:

Script Type	File Type	Extension	Platform
AppleScript	compiled script OSAS file	.scpt (none)	Mac OS
JavaScript ExtendScript	text	.js .jsx	Mac OS & Windows
VBScript	text	.vbs	Windows

Viewing sample scripts

Adobe provides sample scripts for many objects, properties, and methods in the Illustrator CS3 DOM. You can view script samples in two locations:

- In the `/Scripting/Sample Scripts` folder in your Illustrator CS3 installation directory
- In the Adobe Illustrator CS3 scripting reference for your scripting language, which is located in the `/Scripting/Documentation` folder in your Illustratator CS3 installation directory.

Viewing the object model

The supported scripting languages each provide a facility for viewing the scripting objects defined by Illustrator, with reference details.

Viewing the JavaScript object model

To view the JavaScript object model for Illustrator:

1. Start the ExtendScript Toolkit (ESTK).

   Note: In a default Adobe installation, the ESTK is in the following location:

   - In Mac OS:

     system drive`:Applications:Utilities:Adobe Utilities:`
     `ExtendScript Toolkit 2`

   - In Windows:

     system drive`\Program Files\Adobe\Adobe Utilities\`
     `ExtendScript Toolkit 2`

2. In the ESTK, choose Help > Illustrator CS3.

Note: Several extended sample scripts are available in the `/Scripting/Sample Scripts` folder in your Illustrator CS3 installation directory.

You can also view script samples and information about individual classes, objects, properties, methods and parameters in the *Adobe Illustrator CS3 JavaScript Reference*, which is found in the `/Scripting/Documentation` folder in your Illustrator CS3 installation directory.

Viewing the AppleScript object model

Apple provides a Script Editor with all Mac OS systems. You can use Script Editor to view the AppleScript dictionary that describes Illustrator objects and commands.

Note: For details of how to use the Script Editor, see Script Editor Help.

1. Start Script Editor.

Note: In a default Mac OS installation, Script Editor is in Applications:AppleScript:Script Editor. If you cannot find the Script Editor application, you must reinstall it from your Mac OS system CD.

2. Choose File > Open Dictionary. Script Editor displays an Open File dialog.
3. In the Open File dialog, find and select Illustrator, and then click OK.

   Script Editor displays a list of the Illustrator objects and commands, which includes the properties and elements associated with each object and the parameters for each command.

Note: Several extended sample scripts are available in the `:Scripting:Sample Scripts` folder in your Illustrator CS3 installation directory.

You can also view script samples and information about individual classes, objects, properties, methods and parameters in the *Adobe Illustrator CS3 AppleScript Reference*, which is found in the `:Scripting:Documentation` folder in your Illustrator CS3 installation directory.

Viewing the VBScript object model

VBScript provides a type library that you can use to view Illustrator object properties and methods. This procedure explains how to view the type library through any Microsoft Office program. Your VBScript editor most likely provides access to the library. Consult your editor's Help for information.

1. In any Microsoft Office application, choose Tools > Macro > Visual Basic Editor.
2. In the Visual Basic Editor, choose Tools > References.
3. In the dialog that appears, select the check box for Adobe Illustrator CS3 Type Library, and then click OK.
4. Turn on the Adobe Illustrator CS3 Type Library option from the list of available references and click OK.
5. Choose View > Object Browser to display the Object Browser window.
6. Choose "Illustrator" from the list of open libraries in the top-left pull-down menu of the Object Browser window.

Note: Several extended sample scripts are available in the `/Scripting/Sample Scripts` folder in your Illustrator CS3 installation directory.

You can also view script samples and information about individual classes, objects, properties, methods and parameters in the *Adobe Illustrator CS3 VBScript Reference*, which is found in the `/Scripting/Documentation` folder in your Illustrator CS3 installation directory.

Executing scripts

The Illustrator interface includes a Scripts menu (File > Scripts) that provides quick and easy access to your scripts.

- Scripts can be listed directly as menu items that run when you select them. See "Installing scripts in the Scripts menu" on page 10.
- You can also navigate from the menu to any script in your file system, and then run the script. See "Executing scripts from the Other Scripts menu item" on page 10.

You can also have JavaScript scripts with a `.jsx` extension start automatically when you launch the application. For information, see "Startup scripts (.jsx scripts only)" on page 10.

Installing scripts in the Scripts menu

To include a script in the Scripts menu (File > Scripts), save the script in the Scripts folder, which is located in the /Illustrator CS3/ Presets folder in your Illustrator CS3 installation directory. The script's file name, minus the file extension, appears in the Scripts menu.

Note: Scripts that you add to the Scripts folder while Illustrator is running do not appear in the Scripts menu until the next time you launch Illustrator.

Any number of scripts can be installed in the Scripts menu. If you have a large collection of scripts, use subfolders in the Scripts folder to help organize the scripts in the Scripts menu. Each subfolder is displayed as a separate submenu containing the scripts in that subfolder.

Executing scripts from the Other Scripts menu item

The Other Scripts item at the end of the Scripts menu (File > Scripts > Other Scripts) allows you to execute scripts that are not installed in the Scripts folder.

Selecting Other Scripts displays a Browse dialog, which you use to navigate to a script file. When you select the file, the script is executed.

Note: Only files that are of one of the supported file types are displayed in the browse dialog. For information, see "Script support in Adobe Illustrator CS3" on page 7.

Startup scripts (.jsx scripts only)

JavaScript scripts with a `.jsx` file extension can be installed in one of two folders so that the scripts run automatically when you launch Illustrator and each time you run a script. The folders are:

- An application-specific startup scripts folder, which contains scripts for Illustrator CS3
- A general startup scripts folder, which contains scripts that run automatically when you start any Creative Suite 3 application

Application-specific startup scripts folder

You must place application-specific startup scripts in a folder named `Startup Scripts`, which you create in the Illustrator installation directory.

For example, when Illustrator CS3 is installed to its default location, you would create the `Startup Scripts` folder at:

- In Windows:

  `C:\Program Files\Adobe\Adobe Illustrator CS3\Startup Scripts\`

- In Mac OS:

  `/Applications/Adobe Illustrator CS3/Startup Scripts/`

Note: JavaScript scripts with a `.jsx` extension placed in the `Startup Scripts` folder run automatically when:

- The application is launched.
- Any JavaScript file is selected from the Scripts menu (File > Scripts).

General startup scripts folder

The general startup scripts folder contains scripts that run automatically when you start any Creative Suite 3 application. You create the folder in the following location:

- In Windows:

  `Program Files/Common Files/Adobe/Startup Scripts CS3/Illustrator`

- In Mac OS:

  `:Library:Application Support:Adobe:Startup Scripts CS3:Illustrator`

Note: If a script in the general startup folder is meant to be executed only by Illustrator, the script must include the ExtendScript `#target` directive (`#target illustrator`) or code such as the following:

```
if( BridgeTalk.appName == "illustrator" ) {
//continue executing script
}
```

For additional details, see the *JavaScript Tools Guide*.

Changes since earlier versions

The following changes have been made to the scripting object model to support features in Adobe Illustrator CS3:

- Create documents using the `document preset` object, which contains properties for *color mode*, *height* and *width*, *preview mode*, *title*, and other document-defining characteristics:
  - JavaScript: `app.documents.addDocument()`, uses new object `documentPreset`; `app.startupPresetsList` gives available presets; `app.getPresetSettings()` returns a `documentPreset` object.
  - VBScript: `App.Documents.AddDocument`, uses new object `DocumentPreset`; `App.StartupPresetsList` gives available presets; `App.GetPresetSettings` returns a `DocumentPreset` object.
  - AppleScript: `add document` command, uses new object `document preset`; the `startup presets` property gives available presets, `get preset settings` command returns a `document preset` object.
- Undo and redo operations:

- JavaScript: `app.undo()`, `app.redo()`
- VBScript: `App.Undo()`, `App.Redo()`
- AppleScript: `undo` and `redo` commands

- Clipboard operations, previously available only in AppleScript, are supported in VBScript and JavaScript. These methods belong to the `application` object but act upon the current selection:
  - JavaScript: `app.cut()`, `app.copy()`, `app.paste()`
  - VBScript: `App.Cut`, `App.Copy`, `App.Paste`
- Additional control of color:
  - JavaScript: `app.loadColorSettings()`, which loads color settings from a specified file; `app.colorSettingsList` gives available files.
  - VBScript: `App.LoadColorSettings`; `App.ColorSettingsList` gives available files.
  - AppleScript: `load color settings` command; the `color settings` property gives available files.
- New properties for `raster item` objects
  - JavaScript: `rasterItem.overprint`, `rasterItem.colorizedGrayscale`, `rasterItem.transparent`, `rasterItem.channels`, `rasterItem.bitsPerChannel`, `rasterItem.colorants`
  - VBScript: `RasterItem.Overprint`, `RasterItem.ColorizedGrayscale`, `RasterItem.Transparent`, `RasterItem.Channels`, `RasterItem.BitsPerChannel`, `RasterItem.Colorants`
  - AppleScript: `bits per channel`, `channels`, `colorants`, `colorized`, `overprint`, `transparent`
- Capturing clipping-area content to a raster file:
  - JavaScript: `document.imageCapture()`; uses new object `imageCaptureOptions`
  - VBScript: `Document.ImageCapture`; uses new object `ImageCaptureOptions`
  - AppleScript: `image capture` command, uses new object `image capture options`
- Merging of graphic styles:
  - JavaScript: `graphicStyle.mergeTo()`
  - VBScript: `GraphicStyle.MergeTo`
  - AppleScript: `merge` command
- Support for indicating the length (in points) of a path:
  - JavaScript: `pathItem.length`
  - VBScript: `PathItem.Length`
  - AppleScript: `length` of `path item`
- XMP metadata, which names the XMP metadata packet associated with a document:
  - JavaScript: `document.XMPString`
  - VBScript: `Document.XMPString`
  - AppleScript: `XMP string`
- Additional support of Adobe Photoshop® files:

  - JavaScript: `openOptionsPhotoshop.preserveHiddenLayers`, `openOptionsPhotoshop.layerComp`
  - VBScript: `OpenOptionsPhotoshop.PreserveHiddenLayers`, `OpenOptionsPhotoshop.LayerComp`
  - AppleScript: properties of `Photoshop options`: `preserve hidden layers`, `layer comp`
- New method or command for obtaining detailed PPD file information:
  - JavaScript: `app.getPPDFileInfo()` (also, case usage in `printerInfo.pPDInfo` changed to `printerInfo.PPDInfo`)
  - VBScript: `App.GetPPDFileInfo`
  - AppleScript: `get PPD info` command
- Additional support for exporting to Flash SWF format:
  - JavaScript: additional properties for `exportOptionsFlash`
  - VBScript: additional properties for `ExportOptionsFlash`
  - AppleScript: additional properties of `Flash export options`
- AutoCAD support:
  - JavaScript: `preferences.AutoCADFileOptions`, new object `AutoCADFileOptions`
  - VBScript: `Preferences.AutoCADFileOptions`, new object `AutoCADFileOptions`
  - AppleScript: `AutoCAD file options` of `Illustrator preferences`, new object `AutoCAD options`
- Expanded support for clipping artwork when exporting to a Flash file allow you to specify whether to clip the art object, the artboard, or the crop area:
  - JavaScript: `exportOptionsGIF.artClipping` (replaces `exportOptionsGIF.artBoardClipping`)
  - VBScript: `ExportOptionsGIF.ArtClipping` (replaces `ExportOptionsGIF.ArtBoardClipping`)
  - AppleScript: `art clipping` (replaces `artboard clipping`)
- New tracing options property allows you to ignore white fill color.
  - JavaScript: `tracingOptions.ignoreWhite`
  - VBScript: `TracingOptions.IgnoreWhite`
  - AppleScript: `ignore white`

Known Issues

- Scripts that create, save and then close a large number of Illustrator files should periodically quit and relaunch Illustrator. The recommended maximum number of files to process before quitting and relaunching Illustrator is:
  - In Windows: 500 files
  - In Mac OS: 1000 files

  Note: For information on quitting and relaunching Illustrator, see "Launching and activating Illustrator" on page 23 and "Quitting Illustrator" on page 24.

- The "An Illustrator error occurred: 1346458189 ("PARM")" alert may be popped when badly written scripts are repeatedly run in Illustrator from the ExtendScript Toolkit.

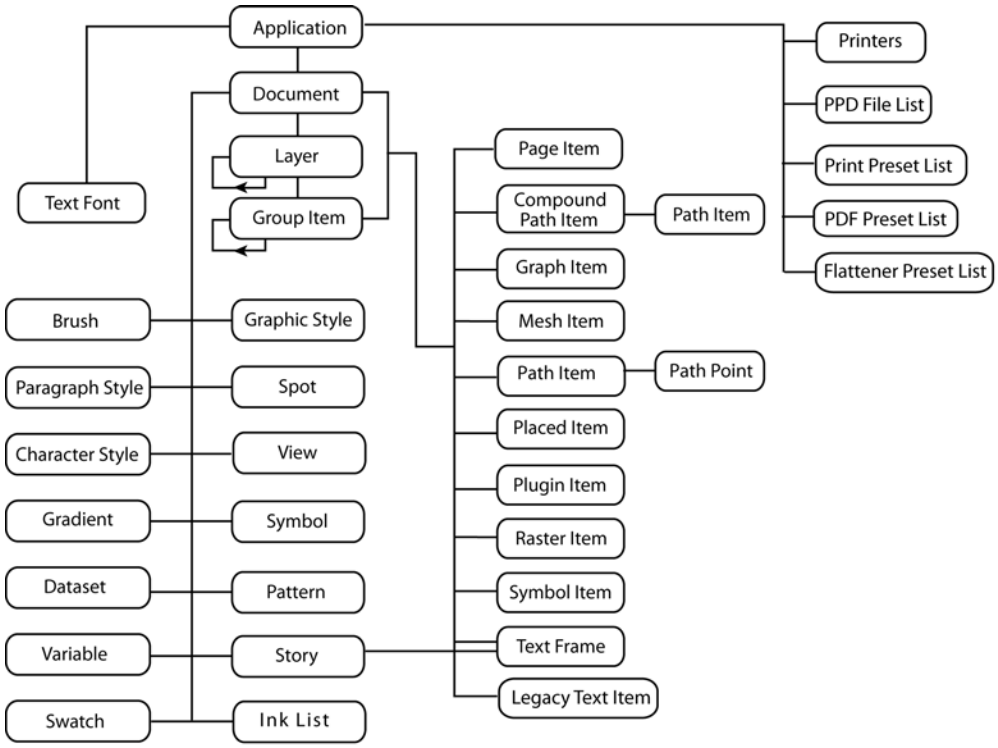
  Scripters need to be very careful about variable initialization and namespace conflict when pushing a batch of Illustrator scripts over and over again for execution in Illustrator via the ExtendScript Toolkit (ESTK) in a single Illustrator session. Each script run is executed within the same persistent ExtendScript engine within Illustrator.

  The ESTK debugger uses BridgeTalk to communicate with Illustrator. A single global, persistent ExtendScript engine inside Illustrator handles all BridgeTalk communications. The net effect is that the state of the ExtendScript engine is cumulative to all the scripts that ran previously. Issues with script code that may cause this problem are:

  - Reading uninitialized variables.
  - Global namespace conflicts, such as when two globals from differents scripts are clobbering each other.

2 The Illustrator Scripting Object Model

A good understanding of the Illustrator object model will improve your scripting abilities. The figure below shows the containment hierarchy of the object model, starting with the `application` object. Note that the `layer` and `group item` classes can contain nested objects of the same class which can, in turn, contain additional nested objects.



Illustrator scripting object model

Note: In addition to this application-specific object model, JavaScript provides certain utility objects, such as the `File` and `Folder` objects, which give you operating-system-independent access to the file system. For details, see the *JavaScript Tools Guide*.

Object naming conventions

There is a single object model for the Illustrator scripting interface, but the actual object names vary slightly in the different scripting languages:

- AppleScript names are all lower case, and individual words are separated by a space. For example:

  ```
  graphic style
  ```

- VBScript names are capitalized, and additional words in the name are indicated by uppercase initial letters. For example:

  ```
  GraphicStyle
  ```

- JavaScript names begin with lowercase letters, and additional words in the name are indicated by uppercase initial letters. For example:

  ```
  graphicStyle
  ```

This chapter uses generic object and property names, but you can easily apply these conventions to determine the corresponding language-specific names.

Note: In the following sections, properties and methods are displayed in *italics*. Object names are displayed in `courier` font.

Top-level (containing) objects

Use these objects to access global information about the Illustrator application or an individual document.

Application

The properties of the `application` object give your script access to global values, such as:

- User `preferences`, which a user sets interactively in the Illustrator application by using the Preferences dialog (Edit > Preferences)
- System information such as installed fonts (the *text fonts* property) and printers (the *printer list* property)

Additionally, there are properties that provide application-specific information and higher-level information about any open documents.

- Application information such as the installation *path*, the *version*, and whether Illustrator is currently *visible*
- The current *active document*; that is, the art canvas that is currently displayed and accepting user input
- All open `documents`

`application` object methods or commands allow your script to perform application-wide actions. For example:

- *Open* files
- *Undo* and *redo* transactions
- *Quit* Illustrator

Document

The `document` object, which your scripts can create or access through the `application` object, represents an art canvas or loaded Illustrator file. The `document` object properties give you access to the document's content. For example:

- The current *selection*, or art objects that the user has selected in the document
- All of the contained art objects, called *page items*, that make up the artwork tree
- Art objects of particular types, such as *symbols* and *text frames*
- All of the *layers*, and the currently *active layer*

Document properties also tell you about the state of the document itself; for example:

- User settings for the document such as *ruler units*
- Whether the document has been *saved* since the last alteration of content
- The *path* of the associated file

The `document` object methods allow your scripts to act on the document. For example:

- *Save* to an Illustrator file or *save as* the various supported file formats
- *Activate* or *close* a document
- *Print* the document; your scripts can select a printer by referencing a `print options` object or reference available printers through the application object's *printer list* property

Layer

The `layer` object provides access to the contents, or artwork tree, of a specific layer. You access the `layer` object through the `document` object. The `layer` object properties provide access to, or information about, the layer, such as:

- Whether the layer is *visible* or *locked*
- The layer *opacity* (overall transparency) and *z order position* (position in the stacking order)
- Art creation preferences for the layer, such as *artwork knockout* and *blending mode*

The artwork tree

The content of an Illustrator document is called the artwork tree. Artwork is represented by the following objects:

- The `compound path item` object
- The `graph item` object
- The `legacy text item` object
- The `mesh item` object
- The `path item` object
- The `placed item` object
- The `plugin item` object
- The `raster item` object
- The `symbol item` object (See "Dynamic objects and symbols" on page 21.)
- The `text frame` object

Your scripts can access and manipulate art objects through collections in the `document` and `layer` objects. There are two types of art object collections:

- Collection objects that correspond to each individual artwork object type, such as the `graph items` object or the `mesh items` object.
- The `page items` object, which includes art objects of all types.

  Additionally, you can use the `group item` object to reference a grouped set of art items.

You can create new art objects using the *make* command (AppleScript) or *add* method of an artwork item collection object. For example, to create a new `path item` object:

AS

```
set myPathItem to make new path item in current document
```

JS

```
var myPathItem = activeDocument.pathItems.add();
```

VB

```
Set myPathItem = appRef.PathItems.Add()
```

Artwork collections that do not allow the creation of new objects using the `make` command or `add` method are:

- The `graph items` object
- The `mesh items` object
- The `plugin items` object
- The `legacy text items` object

For specific information on creating objects of these types, refer to the Adobe Illustrator CS3 scripting references.

Art styles

Your script can apply a graphic style to artwork using the `graphic style` object. To apply a graphic style, you use the *graphic styles* property of the `document` object to access the *apply to* method of the `graphic style` object.

Similarly, the `brush` object allows you to specify the brush to apply to artwork. You access any brush through the `brushes` collection object, which is a property of the `document` object.

Color objects

Your script can apply a color, pattern or gradient to a `path item` object using the *fill color* or *stroke color* properties.

- Scripts can define new color swatches using the *make* command or *add* method of the `swatches` object. Your script can also create a new spot color using the *make* command or *add* property of the `spots` object.
- You can define the attributes of an `ink` object using the `ink info` object, which is an `ink` object property. You access `ink` objects through the *ink list* property of the `document` object.

The following objects allow you to create colors within defined color spaces.

- The `RGB color` object, using the range 0.0 to 255.0 for the each of the three individual color values.
- The `CMYK color` object, using the percentage values 0.0 through 100.0 for each of the four individual color values.

- The `grayscale color` or `LAB color` objects, using the same range and number of values that you use in the Illustrator application.

Text objects

When you type content in an Illustrator document, the type automatically becomes a `text frame` object and, at the same time, a `story` object.

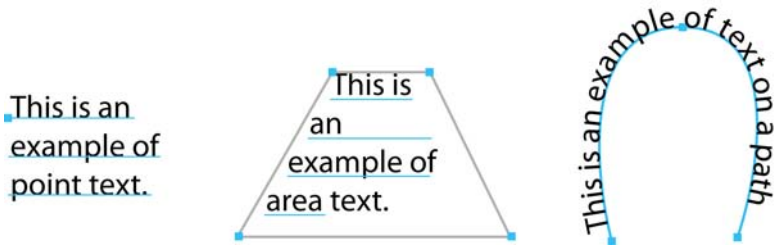
Note: To observe this, open a new document in Illustrator and use the horizontal text tool to type some text, then use the vertical text tool to type some more text, and then create a rectangle and type some text inside it. Now run the following JavaScript script:

```
var myDoc = app.activeDocument
alert("There are " + myDoc.textFrames.length + " text frames.")
alert("There are " + myDoc.stories.length + " stories.")
```

Text frames

A text frame can be one of three kinds:

- *point*
- *area*
- *path*



To create a specific kind of text frame, you use the *kind* property of the `text frames` object in AppleScript. However, the JavaScript and VBScript `text frames` objects contain specific methods for creating area text frames and path text frames.

As in the Illustrator application, you can thread area or path text frames.

To thread existing text frames, you use the *next frame* or *previous frame* property of the `text frame` object. Threaded frames make a single `story` object.

For information on creating or threading text frames, see the chapter for your scripting language.

Text geometry

While the three kinds of text frame have common characteristics, such as an *orientation*, each has type-specific qualities, as reflected in the `text frame` object's properties. For example:

- An area text frame can have rows and columns, which you access through the *row count* and *column count* properties.

- Path text has *start T value* and *end T value* properties that indicate where on the path the text begins and ends.
- Area and path text frames are associated with a `text path` object, which is specified using the `text frame` object's *text path* property. The text path defines the text frame's position and its orientation (horizontal or vertical) on the artboard (while the text frame object's *orientation* property defines the orientation of text within the text frame)

  The *text path* property is not valid for point text, because point text position and orientation are defined completely by the properties of the text frame itself.

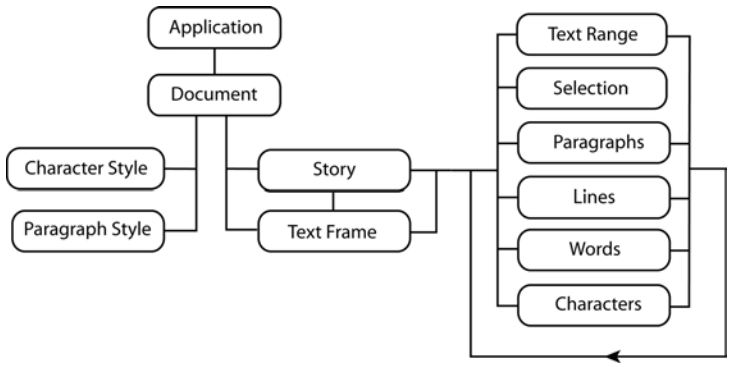
  Note: A text path is not the same as a path art item. Text paths are associated with path art items that can be accessed and manipulated to modify the appearance of the associated text frame.

Objects that represent text content

Within a text frame or story, the actual text content can be accessed as any of the following objects:

- `characters`
- `words`
- `paragraphs`
- `lines`

A `line` object is all of the characters that fit on a single line in a `text frame` or `story` object. All text art items have at least one line of text, defined as a `line` object. Text art can have multiple text lines if the text contains hard line breaks or its characters flow to a new line because they do not fit in the width of the text art. Text objects are accessed and identified by collections within the `text frame` and `story` objects; for example, `textFrame("My Text Frame").paragraphs` or `story("My Story").paragraphs`.



Both `text frame` and `story` objects have *insertion point* and *text selection* properties. The `text frame` object's properties also include the defining features of the text frame, such as:

- The frame *width*, *height*, and *position*
- Whether the frame is *hidden* or *locked*
- Whether the text is *editable*

Note: A `line` object cannot be created in a script. Your script can create `character`, `paragraph`, and `word` objects.

Text ranges

The various text objects within a text frame or story are also represented collectively by the `text range` object. For example, a character is a text range with a length of 1, and a word is a text range that has a space before it.

You can set the content of a `text range` object by passing a string using the *contents* property.

Text styles

Text style elements, such as *font*, *capitalization*, or *justification*, are represented by `paragraph attribute` and `character attribute` objects. These attribute objects are properties of the `paragraph style` and `character style` objects. The `paragraph style` and `character style` objects have *apply to* and *remove* methods that allow your script to assign or remove attributes in a specific paragraph, character, or text range.

You can change the display properties of a text range by applying an appropriate style or providing local overrides of attributes at the text or paragraph levels.

- `character style` objects apply to sets of one or more characters, and control character features such as *font*, *alignment*, *leading*, *language*, and *capitalization*, which are all properties of the `character attribute` object.
- `paragraph style` objects apply to paragraphs, and control paragraph features such as *first line indent*, *left indent*, or *right indent*, which are all properties of the `paragraph attribute` object.

Dynamic objects and symbols

By creating dynamic objects, you can create data-driven graphics. In the Illustrator application, you use the Variables palette to create or edit variables such as graph data, linked file, text string, and visibility variables or variables whose type is not specified. In scripting, you use the `variable` object to represent this type of variable. The `variable` object's *kind* property indicates the type of dynamic data that a `variable` object holds. `variable` objects are document-level objects; you create them in a `document` object.

Note: Do not confuse `variable` objects with scripting variables. For information on Illustrator variables, dynamic objects, and data-driven graphics, refer to Illustrator Help.

Datasets, which collect variables and their associated dynamic data into a single object, are represented in scripting by the `dataset` object. The `dataset` object provides methods with which to update and delete `dataset` objects in your scripts.

In Illustrator, symbols are art items that are stored in the Symbols palette. Your scripts can create, delete, and duplicate `symbol` objects. When you create `symbol` objects in your script, Illustrator adds them to the Symbols palette for the target document.

A `symbol item` is an instance of a `symbol` object in a document. Each `symbol item` is linked to its symbol definition, so that changing the definition of a symbol updates all instances of the symbol.

Your script can create, delete and duplicate symbol items. Symbol items are Illustrator art items and therefore can be treated in the same way as other art items or page items. You can rotate, resize, select, lock, hide and perform other operations on symbol items.

Transformations

The `matrix` object provides access to the power of geometric transformation matrices. Transformation matrices in Illustrator store the settings of an operation that scales, rotates or moves (translates) an object on a page. Some advantages to using matrices are:

- By storing transformation values in a `matrix` object, you can use the values over and over on different objects in your script.
- By concatenating rotation, translation and/or scaling matrices and applying the resulting matrix, you can perform a large series of geometric transformations using only a single script statement.
- You can invert matrix values.
- You can compare the values of two matrices.

The commands or methods to create, get, invert, compare, or concatenate matrices belong to the `application` object.

The command or method used to apply a matrix is the *transform* command, which belongs to any type of object on which transformations can be performed.

3 Scripting Illustrator

This chapter provides an overview of how to use scripting objects to program Adobe Illustrator CS3. Specific examples for the supported scripting languages are provided in the succeeding chapters.

Launching and quitting Illustrator from a script

Your scripts can control the activation and termination of Illustrator.

Launching and activating Illustrator

JS

Typically, you run JavaScript scripts from the application's Scripts menu (File>Scripts) or startup folder, so there is no need to launch Illustrator from your script.

Information on launching Illustrator in a JavaScript is beyond the scope of this guide. For information, search for "interapplication messaging" or "JavaScript messaging framework" in the *JavaScript Tools Guide*.

AS

In AppleScript, you use a `tell` statement to target Illustrator. The `activate` command activates Illustrator if it is not already active.

```
tell application "Adobe Illustrator"
   activate
end tell
```

VBS

In VBScript, there are several ways to create an instance of Illustrator.

- `CreateObject` launches Illustrator as an invisible application if it is not already running. Note that if Illustrator is launched as an invisible application you have to manually activate the application to make it visible.

  ```
  Set appRef = CreateObject("Illustrator.Application")
  ```

Note: If you have multiple versions of Illustrator installed on the same machine and use the `CreateObject` method to obtain an application reference, using "`Illustrator.Application`" creates a reference to the latest Illustrator version. To specifically target an earlier version, use the numeric version identifier at the end of the string:

- For Illustrator 10, use "`Illustrator.Application.1`"
- For Illustrator CS, use "`Illustrator.Application.2`"
- For Illustrator CS2, use "`Illustrator.Application.3`"
- For Illustrator CS3, use "`Illustrator.Application.4`"

- Use the `New` operator if you have added a reference to the Illustrator type library to the project. For example, the following line creates a new reference to the `Application` object:

```
Set appRef = New Illustrator.Application
```

Quitting Illustrator

JS

In JavaScript use the `app.quit()` method to close the application.

```
app.quit()
```

AS

In AppleScript, you use the `quit` command.

```
tell application "Adobe Illustrator"
   quit
end tell
```

VBS

In VBScript, use the `Application` object's `Quit` method to close the application.

```
Set appRef = CreateObject("Illustrator.Application")
appRef.Quit
```

Working with objects

This section provides general information about working with objects in Illustrator.

Getting the frontmost document or layer

To refer to the selected document, you use the `application` object's *current document* property in AppleScript or the *active document* property in JavaScript or VBScript. Similarly, you can use the `document` object's *current layer* or *active layer* property to refer to the selected layer.

Note: There are other types of "active" or "current" object properties, such as *active dataset* or *active view*. Refer to the scripting reference for your language for details.

Creating new objects

There are a number of objects (besides the `application` object itself) that cannot be obtained from containers or parent objects. Your script must create these objects directly.

The following objects must be created explicitly:

`CMYK color`	`ink`	`printer info`
`document preset`	`ink info`	`print flattener options`
`EPS save options`	`matrix`	`print font options`
`export options flash`	`no color`	`print job options`
`export options GIF`	`open options`	`print options`
`export options JPEG`	`paper info`	`print page marks options`
`export options Photoshop`	`Pattern color`	`print paper options`
`export options PNG8`	`PDF open options`	`print postscript options`
`export options PNG24`	`PDF open options`	`screen`
`export options PS5`	`PDF save options`	`screen spot function`
`export options SVG`	`PPD file`	`RGB color`
`file` [a]	`PPD file info`	`spot color`
`folder`	`print color management options`	
`gradient color`	`print color separation options`	
`gray color`	`print coordinate options`	
`Illustrator save options`	`printer`	

a. `File` and `Folder` objects are Adobe ExtendScript devices designed to provide platform-independent access to the underlying file system. For information on using these objects, see the *JavaScript Tools Guide*.

See the chapter for your scripting language for information on creating an object explicitly.

Collection objects

Most collection objects must be obtained from a container. For example, a `path items` collection object can be contained by a `document` object or a `layer` object. To obtain an object in a `path items` collection, you refer to either containing object.

For example:

AS

To refer to a `path items` object in a document:

```
path item 1 in document 1
```

To refer to a `path items` object in a layer:

```
path item 1 in layer 1 in document 1
```

JS

To refer to a `path items` object in a document:

```
documents[0].pathItems[1]
```

To refer to a `path items` object in a layer:

```
documents[0].layers[0].pathItems[0]
```

VBS

To refer to a `path items` object in a document:

```
Documents(1).PathItems(1)
```

To refer to a `path items` object in a layer:

```
Documents(1).Layers(1).PathItems(1)
```

For more examples of collection item containers, refer either to `document` object **Elements** table in the *Adobe Illustrator CS3 AppleScript Reference*, or the **Properties** table in the *Adobe Illustrator CS3 JavaScript Reference* or *Adobe Illustrator CS3 VBScript Reference*. Also, view a diagram of the Illustrator CS3 object model in "The Illustrator Scripting Object Model" on page 15.

Selected objects

There are times when you want to write scripts that act upon the currently selected object or objects. For example, you might want to apply formatting to selected text or change a selected path's shape.

Selecting text

To select text, you use the *select* command or method of the `text range` object.

Selecting art items

You can select an art object (such as graph items, mesh items, raster items, symbol items, and so on) by setting its *selected* property to `true`. (In AppleScript, *selected* is a property of the `page items` object.)

Referring to selected art items

To refer to all currently selected objects in a document, you use the `document` object's *selection* property. To work with the objects in the selection array, you must determine their type in order to know which properties and methods or commands you can use with them. Each artwork object type has a read-only *typename* property in JavaScript or VBScript that you can use to determine the object's type. In AppleScript, use the *class* property.

Notes on renaming objects stored in the application's palettes

Several objects can be renamed; that is, their *name* property is writeable. The following types of objects can be sorted alphabetically in the corresponding Illustrator palette, so if a script modifies the name of such an object, references to that object by index can become invalid. These object types include the following:

```
Brush
Gradient
Graphic Style
Pattern
Swatch
Symbol
Variable
```

Measurement units

Illustrator uses points as the unit of measurement for almost all distances, where one inch is equal to 72 points. The one exception is the values for properties such as *kerning*, *tracking*, and the *aki* properties (used for Japanese text composition), which use em units. (See "Em space units" on page 27.)

Illustrator uses points when communicating with your scripts **regardless of the current ruler units**. If your script depends on adding, subtracting, multiplying, or dividing specific measurement values for units other than points, it must perform any unit conversions needed to represent your measurements as points.

For example, to use inches for coordinates or measurement units, you must multiply all inch values by 72 when entering the values in your script.

The following table shows the conversion formulae for various units of measurement:

Unit	Conversion formula
centimeters	28.346 points = 1 centimeter
inches	72 points = 1 inch
millimeters	2.834645 points = 1 millimeter
picas	12 points = 1 pica
Qs	0.709 point = 1 Q (1 Q equals 0.23 millimeter)

Note: JavaScript provides the `UnitValue` object type, which offers unit conversion utilities. For details, see the *JavaScript Tools Guide*.

Em space units

Values that use em units instead of points are measured in thousandths of an em unit.

Em units are proportional to the current font size. For example, in a 6-point font, 1 em equals 6 points; in a 10-point font, 1 em equals 10 points. Similarly, a kerning value of 20 em units for a 10-point font would be equivalent to:

```
(20 units x  10 points) / 1000 units/em  =  0.2 points
```

Page item positioning and dimensions

Illustrator uses simple two-dimensional geometry in the form of points to record the *position* of `page item` objects in a document. Every `page item` object in a document has a *position* property that defines a fixed point as a pair of page coordinates in the format [x, y]. The fixed point is the top left corner of the object's bounding box.

Note: See "The artwork tree" on page 17 for information on the types of objects that comprise the `page items` collection.

A point is designated by a pair of coordinates:

- The horizontal position *x*
- The vertical position *y*

Note: You can see these coordinates in the Info palette when you select or create an object in Illustrator.

The default *ruler origin* point (0, 0) for coordinate numbering in Illustrator is the lower left corner of the document. (*ruler origin* is a property of the `document` object.)

- On the horizontal axis, coordinates to the right of the ruler's zero point are positive numbers.
- On the vertical axis, coordinates above the zero point are positive.

The default *page origin* property of a `document` object defines the lower left corner of the printable region of the document as a fixed point.

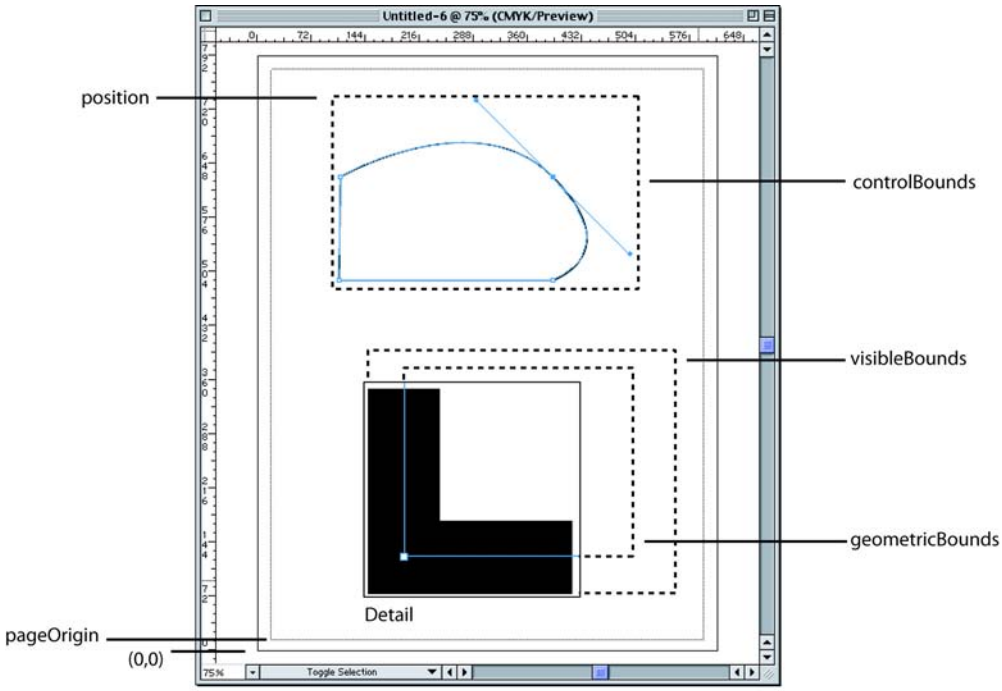
Additionally, each `page item` object has a *width* and *height* property. The maximum value allowed for the width or height of a page item is 16348 points.

Art item bounds

Every `page item` object also has three properties that use fixed rectangles to describe the object's overall extent.

- The *geometric bounds* of a page item are the rectangular dimensions of the object's bounding box excluding stroke width.
- The *visible bounds* of a page item are the dimensions of the object including any stroke widths.
- The *control bounds* define the rectangular dimensions of the object including in- and out- control points.

The following figure illustrates these properties, using the JavaScript naming convention.



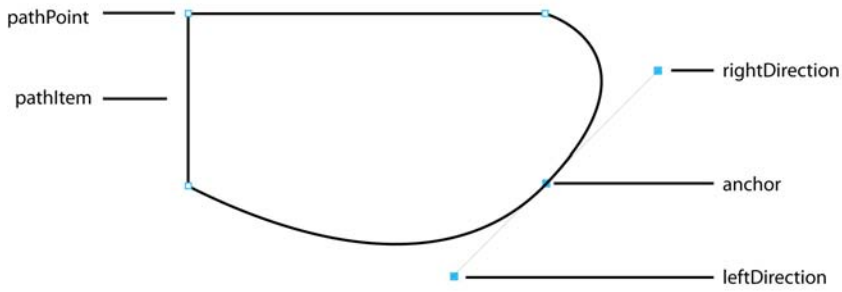
Paths and shapes

Paths are represented in the Illustrator DOM by the `path item` object. Path items include all artwork that contains paths, such as rectangles, ellipses, and polygons, as well as freeform paths.

For information on creating shapes, please see the chapter for your scripting language.

A freeform path consists of a series of path points. A path point can be specified in two ways:

- As an array of x and y page coordinates.
- As a `path point` object, which defines an anchor point and two direction points or handles that define the path segment's curve.



For specific details and samples, see the chapter for your scripting language.

User interaction levels

An application typically presents a dialog when it user feedback is required. This is called user interaction, and is useful and expected when you are directly interacting with the application. However, when a script is interacting with an application, a dialog brings the execution of the script to a halt until the dialog is dismissed. This can be a serious problem in an automation environment where there is no one present to deal with dialogs.

The `application` object contains a *user interaction level* property that allows you to control the level of interaction allowed during script execution. You can suppress interaction in an automation environment, or allow some interaction where scripts are being used in a more interactive fashion.

JS

There are two possible values for the `app.userInteractionLevel` property in JavaScript:

Property Value	Result
`DISPLAYALERTS`	Interaction is allowed
`DONTDISPLAYALERTS`	No interaction is allowed

VBS

There are two possible values for the `UserInteractionLevel` property of the `Application` object in VBScript:

Property Value	Result
`aiDisplayAlerts`	Interaction is allowed
`aiDontDisplayAlerts`	No interaction is allowed

AS

Using AppleScript, it is possible to send commands from one machine to another, so additional types of interaction are possible. There are four possible values for the user interaction level property in AppleScript:

Property Value	Result
`never interact`	No interaction is allowed.
`interact with self`	Interact only with scripts executed from the Scripts menu (File > Scripts).
`interact with local`	Interact with scripts executed on the local machine (including self).
`interact with all`	Interact with all scripts.

The four values allow you to control interaction based on the source of the script commands. For example, if the application is acting as a server for remote users, it would be difficult for a remote user to dismiss a dialog, but it would be no problem for someone sitting in front of the machine. In this case, an interaction level of *interact with local* would prevent dialogs from halting remote scripts but would allow dialogs to be presented for local scripts.

Printing Illustrator documents

Using the `print options` scripting feature, you can capture and automate parts of your print workflow. Scripting exposes the full capabilities of Illustrator printing, some of which may not be accessible through the application's user interface.

Note: Illustrator supports at most one print session at any give time because of limitations in the current printing architecture.

The `document` object's *print* command or method takes a single optional parameter, which allows you to specify a `print options` object.

The `print options` object allows you to define print settings such as PPD, PostScript options, paper options, color management options, and so on. The `print options` object also has a *print preset* property, which allows you to specify a preset to define your print job.

When defining the properties of a `print options` object, you can find out which printers, PPDs, print presets, and other items are available by using the `application` object's read-only "list" properties, such as the *printer list* property, the *PPD file list* property, the *print presets list* property, and so on.

4 Scripting with JavaScript

This chapter uses script examples and explanations to help you to become familiar with Illustrator scripting using JavaScript.

For more information

Several extended sample scripts are available in the `/Scripting/Sample Scripts` folder in your Illustrator CS3 installation directory.

For information about individual classes, objects, properties, methods and parameters, as well as script samples that demonstrate how to use many of them, refer to the *Adobe Illustrator CS3 JavaScript Reference*, which is found in the `/Scripting/Documentation` folder in your Illustrator CS3 installation directory. You can also use the Illustrator dictionary, which you access from the Object Model Viewer in the ExtendScript Toolkit. For information on using the ExtendScript Toolkit and the Object Model Viewer, see "Viewing the JavaScript object model" on page 8 or the *JavaScript Tools Guid*e.

If you are a beginner and find that you don't understand the concepts and terms used in this chapter, read the *Adobe Introduction to Scripting*.

Your first Illustrator script

The traditional first project in any programming language is to display the message "Hello World!" In this example, you create a new Illustrator document, then add a text frame containing this message.

1. Using any text editor (including Adobe InDesign® software or the ExtendScript Tookit), enter the following text:

   Note: For information on locating the ExtendScript Toolkit, see "Viewing the JavaScript object model" on page 8.

   ```
   //Hello World!
   var myDocument = app.documents.add();
   //Create a new text frame and assign it to the variable "myTextFrame"
   var myTextFrame = myDocument.textFrames.add();
   // Set the contents and position of the text frame
   myTextFrame.position = [200,200];
   myTextFrame.contents = "Hello World!"
   ```

2. To test the script, do either of the following:
   - If you are using the ExtendScript Toolkit, select Illustrator CS3 from the dropdown list in the upper left corner and select Yes to start Illustrator, and then choose Debug> Run in the ExtendScript Toolkit to run the script.
   - If you are using a different text editor, save the file as text only in a folder of your choice, using the file extension `.jsx`, and then start Illustrator. In Illustrator, choose File>Scripts>Other Scripts, and then navigate to and run your script file.

Tip: To add the script to the Illustrator Scripts menu (File > Scripts), save the script in the Scripts folder. The script will appear on the menu the next time you start Illustrator. For information, see "Installing scripts in the Scripts menu" on page 10.

Adding features to "Hello World"

Next, let's create a new script that makes changes to the Illustrator document you created with your first script.

Our second script will demonstrate how to:

- Get the active document.
- Get the width of the active document.
- Resize the text frame to match the document's width.

Note: If you've already closed the Illustrator document, run your first script again to create a new document before you proceed with this exercise.

1. Choose File > New in your text editor to create a new script.
2. Enter the following code:

```
var docRef = app.activeDocument;
var docWidth = docRef.width
var frameRef = docRef.textFrames[0]
frameRef.width = docWidth
```

3. Run the script.

Working with methods in JavaScript

When you work with methods that have multiple parameters, you may omit optional parameters at the end of the parameter list, but you may not omit parameters in the middle of the list. If you do not wish to specify a particular parameter in the middle of the list, you must insert the value `undefined` to use the parameter's default value. For example, the following definition describes the `rotate()` method for an art object.

Note: In the definition, taken from the *Adobe Illustrator CS3 JavaScript Reference*, optional parameters are enclosed in square brackets (`[]`).

```
rotate
   (angle
   [,changePositions]
   [,changeFillPatterns]
   [,changeFillGradients]
   [,changeStrokePattern]
   [,rotateAbout])
```

To rotate the object 30 degrees and change the `fillGradients`, you would use the following script statement:

```
myObject.rotate(30, undefined, undefined, true);
```

Note that you only need to specify `undefined` for the `changePositions` and `changeFillPatterns` parameters. You do not have to specify anything for optional parameters that follow the `changeFillGradients` parameter.

Accessing and referencing objects

When you write a script, you must first decide which file, or `document`, the script should act on. Through the `application` object, the script can create a new document, open an existing document, or act on a document that is already open.

The script can create new objects in the document, operate on objects that the user has selected, or operate on objects in one of the object collections. The following sections illustrate various techniques for accessing, referencing, and manipulating Illustrator objects.

Referencing the application object

To obtain a reference to a specific object, you need to navigate the containment hierarchy. However, because all JavaScript scripts are executed from within the Illustrator application, a specific reference to the `application` object is not required. For example, to assign the active document in Illustrator to the variable `frontMostDocument`, you could reference the `activeDocument` property of the `application` object as follows:

```
var frontMostDocument = activeDocument;
```

It is permissible to use the `application` object in a reference. To reference the `application` object, you use the global variable `app`. The following two statements appear identical to the JavaScript engine:

```
var frontMostDocument = activeDocument;

var frontMostDocument = app.activeDocument;
```

Accessing objects in collections

All open documents, as well as the objects in a document, are collected into collection objects for the object type. A collection object contains an array of the objects that you can access by index or by name. The collection object takes the plural form of the object name. For example, the collection object for the `document` object is the `documents` object.

The following script sample gets all `graphic style` objects in the graphic styles collection; that is, it gets all graphic styles available to the active document:

```
var myStyles = app.activeDocument.graphicstyles;
```

All numeric collection references in JavaScript are zero-based; that is, the first object in the collection has the index [0].

Note: As a rule, JavaScript index numbers do not shift when you add an object to a collection. However, there is one exception: `documents[0]` is always the active or frontmost document.

To access the first style in graphic styles collection, you can use the variable declared in the previous script sample, or you can use the containment hierarchy to refer to the collection:

- Using the `myStyles` variable:

```
var firstStyle = myStyles[0];
```

- Using the containment hierarchy:

```
var firstStyle = app.activeDocument.graphicStyles[0];
```

The following statements assign the name "Modern" to the first graphic style in the collection. These statements are identical; you can use them interchangeably.

```
myStyles[0].name = 'Modern'

firstStyle.name = 'Modern'

app.activeDocument.graphicStyles[0].name = 'Modern'
```

To get the total number of objects in a collection, use the `length` property:

```
alert ( myStyles.length ) ;
```

The index of the last graphic style in the collection would be `myStyles.length-1` (-1 because the collection starts the index count at 0 and the `length` property counts from 1):

```
var lastStyle = myStyles[ myStyles.length - 1 ];
```

Notice that an expression representing the index value is enclosed in square brackets (`[]`) as well as quotes.

If you know the name of an object, you can access the object in the collections using the name surrounded by square brackets. For example:

```
var getStyle = myStyles['Ice Type'];
```

Notice that the object name is enclosed in square brackets (`[]`).

Each element in the collection is an object of the desired type, and you can access its properties through the collection. To get an object's name, for example, use the `name` property:

```
var styleName = app.activeDocument.graphicStyles[0].name;
```

To apply `lastStyle` to the first `pageItem` in the document, use its `applyTo()` method:

```
lastStyle.applyTo( app.activeDocument.pageItems[0] );
```

Creating new objects

You can use a script to create new objects. To create objects that are available from collection objects, or *containers*, you use the container object's `add()` method.

```
var myDoc = app.documents.add()
var myLayer = myDoc.layers.add()
```

Some object types are not available from containers. You create an object of this type by defining a variable, and then using the `new` operator with an object constructor to assign an object as the value. For example, to create a new `CMYKColor` object using the variable name `myColor`:

```
var myColor = new CMYKColor()
```

Working with selections

When the user makes a selection in a document, the selected objects are stored in the document's `selection` property. To access all selected objects in the active document:

```
var selectedObjects = app.activeDocument.selection;
```

The `selection` property value can be an array of any type of art objects, depending on what types of objects are selected. To get or manipulate the properties of the selected art items, you must retrieve the individual items in the array. To find out an object's type, use the `typename` property.

The following sample gets the first object in the array, and then displays the object's type:

```
var topObject = app.activeDocument.selectedObjects[0];
alert(topObject.typename)
```

Note: The first object in a selection array is the selected object that was last added to the page, and not the last object selected.

Selecting artwork objects

To select an art object, use the object's `selected` property.

Working with text frames

To create a text frame of a specific type in JavaScript, you use the `textFrames` method whose name corresponds to the text frame type. For example:

```
var rectRef = docRef.pathItems.rectangle(700, 50, 100, 100);

//use the areaText method to create the text frame
var areaTextRef = docRef.textFrames.areaText(rectRef);
```

Threaded frames

As in the Illustrator application, you can thread area or path text frames.

To thread existing text frames, you use the `nextFrame` or `previousFrame` property of the `text frame` object.

Note: When copying the following script to the ExtendScript ToolKit, place the value of the *contents* property a single line.

```
var myDoc = documents.add();
var myPathItem1 = myDoc.pathItems.rectangle(244, 64, 82, 76);
var myTextFrame1 = myDoc.textFrames.areaText(myPathItem1);
var myPathItem2 = myDoc.pathItems.rectangle(144, 144, 42, 116);
var myTextFrame2 = myDoc.textFrames.areaText(myPathItem2);

// use the nextFrame property to thread the text frames
myTextFrame1.nextFrame = myTextFrame2;
var sText = "This is two text frames linked together as one story, with text
   flowing from the first to the last. This is two text frames linked
   together as one story, with text flowing from the first to the last. This
   is two text frames linked together as one story. ";
myTextFrame1.contents = sText;
redraw();
```

Threaded frames make a single story object

Threaded frames make a single `story` object.

To observe this, run the following JavaScript after running the script in "Threaded frames" on page 35.

```
var myDoc = app.activeDocument
alert("There are " + myDoc.textFrames.length + " text frames.")
alert("There are " + myDoc.stories.length + " stories.")
```

Creating paths and shapes

This section explains how to create items that contain paths.

Paths

To create a freeform path, you specify a series of path points, either as series of x-y coordinates or as `pathPoint` objects.

- Using x-y coordinates limits the path to straight segments only.
- To created a curved path, you must create `pathPoint` objects.

Your path can consist of a combination of page coordinates and `pathPoint` objects.

Specifying a series of x-y coordinates

To specify a path using page coordinate pairs, you use the `setEntirePath()` method of the `pathItems` object. The following script specifies three pair of x-y coordinates to create a path that has three points.

```
var myDoc = app.activeDocument;
var myLine = myDoc.pathItems.add();
//set stroked to true so we can see the path
myLine.stroked = true;
myLine.setEntirePath([[220, 475], [375, 300], [200, 300]]);
```

Using pathPoint objects

When you create a `pathPoint` object, you define three values for the point:

- A fixed *anchor* point, which is the point on the path
- A pair of direction points: the *left direction* point and the *right direction* point, which allow you to control the path segment's curve

  You define each property as an array of page coordinates in the format [x, y].

  - If all three properties of a `pathPoint` object have the same coordinates, and the properties of the next `pathPoint` in the line are equal to each other, you create a straight line segment.
  - If two or more properties in a `pathPoint` object hold different values, the segment connected to the point is curved.

To create a path, or to add points to an existing path, using `pathPoint` objects, you create a `pathItem` object and then add the path points as child objects in the `pathItem`.

```
var myDoc = app.activeDocument;
var myLine = myDoc.pathItems.add();
   //set stroked to true so we can see the path
   myLine.stroked = true;

var newPoint = myLine.pathPoints.add();
   newPoint.anchor = [220, 475];
   //giving the direction points the same value as the
   //anchor point creates a straight line segment
   newPoint.leftDirection = newPoint.anchor;
   newPoint.rightDirection = newPoint.anchor;
   newPoint.pointType = PointType.CORNER;

var newPoint1 = myLine.pathPoints.add();
   newPoint1.anchor = [375, 300];
   newPoint1.leftDirection = newPoint1.anchor;
   newPoint1.rightDirection = newPoint1.anchor;
   newPoint1.pointType = PointType.CORNER;

var newPoint2 = myLine.pathPoints.add();
   newPoint2.anchor = [220, 300];
   //giving the direction points different values
   //than the anchor point creates a curve
   newPoint2.leftDirection =[180, 260];
   newPoint2.rightDirection = [240, 320];
   newPoint2.pointType = PointType.CORNER;
```

Combining path point types

The following script sample creates a path with three points:

```
var myDoc = app.activeDocument;
var myLine = myDoc.pathItems.add();
   myLine.stroked = true;
   myLine.setEntirePath( [[220, 475], [375, 300]]);

// Append another point to the line
var newPoint = myDoc.myLine.pathPoints.add();
     newPoint.anchor = [220, 300];
     newPoint.leftDirection = newPoint.anchor;
     newPoint.rightDirection = newPoint.anchor;
     newPoint.pointType = PointType.CORNER;
```

Shapes

To create a shape, you use the `pathItems` method that corresponds to the shape's name (such as *ellipse*, *rectangle*, or *polygon*), and use the parameters to specify shape's position, size, and other information such as the number of sides in a polygon.

Tip: Remember:

- All measurements and page coordinates are processed as points by the scripting engine. For more information, see "Measurement units" on page 27.
- x and y coordinates are measured from the bottom left corner of the document, as indicated in the Info palette in the Illustrator application. For information, see "Page item positioning and dimensions" on page 27.

Creating a rectangle

The following sample uses the `textFrames` object's `rectangle()` method to create a rectangle with the following properties:

- The top of the rectangle is 2 inches (144 points) from the bottom edge of the page.
- The left edge is 2 inches (144 points) from the left edge of the page.
- The rectangle is 1 inch wide and 3 inches long.

```
var myDocument = app.documents.add()
var artLayer = myDocument.layers.add()
var rect = artLayer.pathItems.rectangle( 144, 144, 72, 216 );
```

Creating a polygon

The following sample uses the `polygon()` method to create a polygon with the following properties:

- The center point of the object is inset is 2 inches (144 points) on the horizontal axis and 4 inches (288 points) on the vertical axis.
- The length of the radius from the center point to each corner is 1 inch.
- The polygon has 7 sides.

```
var myDocument = app.documents.add()
var artLayer = myDocument.layers.add()
var poly = artLayer.pathItems.polygon( 144, 288, 72.0, 7 );
```

5 Scripting with AppleScript

This chapter uses script examples and explanations to help you to become familiar with Illustrator scripting using AppleScript.

For more information

Several extended sample scripts are available in the `:Scripting:Sample Scripts` folder in your Illustrator CS3 installation directory.

For information about individual classes, objects, properties, commands and parameters, as well as script samples that demonstrate how to use many of them, refer to the *Adobe Illustrator CS3 AppleScript Reference*, which is found in the `:Scripting:Documentation` folder in your Illustrator CS3 installation directory. You can also use view Illustrator CS3 dictionary from the Script Editor application. See "Viewing the AppleScript object model" on page 9.

If you are a beginner and find that you don't understand the concepts and terms used in this chapter, read the *Adobe Introduction to Scripting*.

Your first Illustrator script

The traditional first project in any programming language is to display the message "Hello World!" In this example, you create a new Illustrator document, then add a text frame containing this message.

1. Open Script Editor.

Note: In a default Mac OS installation, Script Editor is in `Applications:AppleScript:Script Editor`. If you cannot find the Script Editor application, you must reinstall it from your Mac OS system CD.

2. Enter the following script.

```
--Send the following commands to Illustrator
tell application "Adobe Illustrator"
--Create a new document
set docRef to make new document
--Create a new text frame with the string "Hello World"
set textRef to make new text frame in docRef ¬
       with properties {contents: "Hello World!", position:{200, 200}}
end tell
```

3. In the Script Editor toolbar, click Run.

Tip: To add the script to the Illustrator Scripts menu (File > Scripts), save the script in the Scripts folder. The script will appear on the menu the next time you start Illustrator. For information, see "Installing scripts in the Scripts menu" on page 10.

Adding features to "Hello World"

Next, let's create a new script that makes changes to the Illustrator document you created with your first script.

Our second script will demonstrate how to:

- Get the active document.
- Get the width of the active document.
- Resize the text frame to match the document's width.

Note: If you've already closed the Illustrator document, run your first script again to create a new document.

1. Choose File > New in Script Editor to create a new script.
2. Enter the following code:

```
tell application "Adobe Illustrator"
-- current document is always the active document
set docRef to the current document
set docWidth to the width of docRef
-- resize the text frame to match the page width
set width of text frame 1 of docRef to docWidth
-- alternatively, one can reference the item directly, as follows:
set width of text frame 1 of current document to docWidth
end tell
```

3. Run the script.

Object references

In AppleScript, Illustrator returns object references by index position or name. For example, a reference to the first path in layer 2 would be:

```
path item 1 of layer 2 of document 1
```

An object's index position may change when other objects are created or deleted. For example, when a new path item is created on `layer 2`, the new path item will become `path item 1 of layer 2 of document 1`. This new object displaces our original path item, forcing the original to index position 2. Therefore, any references made to `path item 1 of layer 2 of document 1` will refer to the new object. This method of applying index numbers assures that lowest index number refers to the object that has been worked on most recently. Consider the following sample script.

```
-- Make 2 new objects and try to select both
tell application "Adobe Illustrator"
   set newDocument to make new document
   set rectPath to make new rectangle in newDocument
   set starPath to make new star in newDocument
   set selection of newDocument to {rectPath, starPath}
end tell
```

This script does not select both the rectangle and the star, as intended; instead, it selects only the star. Try running the script with the Event Log window open to observe the references returned from Illustrator for each of the consecutive `make` commands (Choose Event Log at the bottom of the Script Editor window).

Notice that both commands return the same object reference: `path item 1 of layer 1 of document 1`. Therefore, the last line resolves to:

```
set selection of document 1 to {path item 1 of layer 1 of document 1, ¬
  path item 1 of layer 1 of document 1}
```

A better approach is to reference the objects by name:

```
tell application "Adobe Illustrator"
   set newDocument to make new document
   make new rectangle in newDocument with properties {name:"rectangle"}
   make new star in newDocument with properties {name:"star"}
   set selection of newDocument to ¬
      {path item "rectangle" of newDocument, ¬
       path item "star" of newDocument}
end tell
```

This example illustrates the need to uniquely identify objects in AppleScript scripts. It is recommended that you assign names or variables to objects you need to access at a later time, as there is no guarantee you are accessing the objects you expect when accessing them by index.

Obtaining objects from documents and layers

This script references an object as part of a document:

```
-- Get reference for first page item of document 1
tell application "Adobe Illustrator"
   set pageItemRef to page item 1 of document 1
end tell
```

In the following script, the variable `pageItemRef` does not necessarily refer to the same object as in the previous script, because this script includes a reference to a layer:

```
-- Get reference for first page item of layer 1 of document 1
tell application "Adobe Illustrator"
   set pageItemRef to page item 1 of layer 1 of document 1
end tell
```

Creating new objects

To create a new object in AppleScript, you use the `make` command.

Working with selections

When the user makes a selection in a document, the selected objects are stored in the document's `selection` property. To access all selected objects in the active document:

```
tell application "Adobe Illustrator"
   set myDoc to current document
   set selectedObjects to selection of myDoc
end tell
```

Depending on what is selected, the `selection` property value can be an array of any type of art objects. To get or manipulate the properties of the selected art items, you must retrieve the individual items in the array. To find out an object's type, use the *class* property.

The following sample gets the first object in the array, and then displays the object's type:

```
tell application "Adobe Illustrator"
   set myDoc to current document
   set selectedObjects to selection of myDoc
   set topObject to item 1 of selectedObjects
   display dialog (class of topObject)
end tell
```

Note: The first object in a selection array is the selected object that was last added to the page, and not the last object selected.

Selecting artwork objects

To select an art object, use the object's `selected` property.

Working with text frames

To create a text frame of a specific type in AppleScript, you use the `kind` property of the `text frame` object.

```
set myRect make new rectangle in current document with properties ¬
   {position:{100, 700}, height:100, width:100}
set myAreaText make new text frame in current document with properties ¬
   {kind:area text,contents:"Text Frame 1"}
```

Threaded frames

As in the Illustrator application, you can thread area or path text frames.

To thread existing text frames, you use the `next frame` or `previous frame` property of the `text frame` object.

Note: When copying the following script to your script editor, place the value of the *contents* property on a single line. The long line character (¬) is not valid within a string value.

```
tell application "Adobe Illustrator"
   make new document
   make new rectangle in current document with properties ¬
      {position:{100, 500}, height:100, width:100}
   make new text frame in current document with properties ¬
      {kind:area text, text path:the result, name:"tf1", ¬
      contents:"This is two text frames linked together as one story, with
      text flowing from the first to the last. First frame content. "}
   make new rectangle in current document with properties ¬
      {position:{300, 700}, height:100, width:100}
   make new text frame in current document with properties ¬
      {kind:area text, text path:the result, name:"tf2", ¬
      contents:"Second frame content." }
   --use the next frame property to thread the frames
      set next frame of text frame "tf1" of current document to ¬
      text frame "tf2" of current document
   redraw
end tell
```

Threaded frames make a single story object

Threaded frames make a single `story` object.

To observe this, run the following JavaScript after running the script in "Threaded frames" on page 43.

```
var myDoc = app.activeDocument
alert("There are " + myDoc.textFrames.length + " text frames.")
alert("There are " + myDoc.stories.length + " stories.")
```

Creating paths and shapes

This section explains how to create items that contain paths.

Paths

To create line or a freeform path, you specify a series of path points, either as series of x-y coordinates or as `path point` objects.

- Using x-y coordinates limits the path to straight segments only.
- To created a curved path, you must create `path point` objects.

A path can consist of a combination of page coordinates and `path point` objects.

Specifying a series of x-y coordinates

To specify a path using page coordinate pairs, you use the `entire path` property of the `path items` object. The following script specifies three pair of x-y coordinates to create a path that has three points.

```
tell application "Adobe Illustrator"
set docRef to make new document
-- set stroked to true so we can see the path
set lineRef to make new path item in docRef with properties {stroked:true}
set entire path of lineRef to {{220, 475},{200, 300},{375, 300}}
end tell
```

Using path point objects

To create a `path point` object, you must define three values for the point:

- A fixed *anchor* point, which is the point on the path
- A pair of direction points: the *left direction* point and the *right direction* point, which allow you to control the path segment's curve

  You define each property as an array of page coordinates in the format [x, y].

  - If all three properties of a `path point` object have the same coordinates, and the properties of the next `path point` in the line are equal to each other, you create a straight line segment.
  - If two or more properties in a `path point` object hold different values, the segment connected to the point is curved.

To create a path, or to add points to an existing path, using `path point` objects, you create a `path item` object and then add the path points as child objects in the `path item`.

```
tell application "Adobe Illustrator"
set docRef to make new document
-- set stroked to true so we can see the path
set lineRef to make new path item in docRef with properties {stroked:true}

   --giving the direction points the same value as the
   --anchor point creates a straight line segment
set newPoint to make new path point of lineRef with properties ¬
   {anchor:{220, 475},left direction:{220, 475},right direction:{220, 475}¬
   point type:corner}

set newPoint2 to make new path point of lineRef with properties ¬
   {anchor:{375, 300},left direction:{375, 300},right direction:{375, 300}¬
   point type:corner}

   --giving the direction points the different values
   --creates a curve
set newPoint3 to make new path point of lineRef with properties ¬
   {anchor:{220, 300},left direction:{180, 260},right direction:{240, 320}¬
   point type:corner}

end tell
```

Combining path point types

The following script sample creates a path with three points by combining the entire path property with a `path point` object:

```
tell application "Adobe Illustrator"
set docRef to make new document
-- set stroked to true so we can see the path
set lineRef to make new path item in docRef with properties {stroked:true}
set entire path of lineRef to {{220, 475},{375, 300}}
set newPoint to make new path point of lineRef with properties ¬
   {anchor:{220, 300},left direction:{180, 260},right direction:{240, 320}¬
   point type:corner}
end tell
```

Shapes

To create a shape, you use the object that corresponds to the shape's name (such as *ellipse*, *rectangle*, or *polygon*), and use the object's properties to specify shape's position, size, and other information such as the number of sides in a polygon.

Tip: Remember:

- All measurements and page coordinates are processed as points by the scripting engine. For more information, see "Measurement units" on page 27.
- x and y coordinates are measured from the bottom left corner of the document, as indicated in the Info palette in the Illustrator application. For information, see "Page item positioning and dimensions" on page 27.

write-once access

Properties for path item shapes use the access status “write-once”, which indicates that the property is writeable only at the time the object is created. For existing path item objects, the properties are read-only properties whose values cannot be changed.

Creating a rectangle

The following sample creates a rectangle with the following properties:

- The top right corner of the of the rectangle is inset 4 inches (288 points) from the bottom of the page and 5 inches (360 points) from the left edge of the page.
- The lower left corner of the rectangle is inset 1 inch (72 points) from the left edge of the page and 2 inches (144 points) from the bottom of the page.

```
tell application "Adobe Illustrator"
set docRef to make new document
set rectRef to make new rectangle in docRef with properties ¬
   {bounds:{288, 360, 72, 144}}
end tell
```

Creating a polygon

The following sample creates a polygon with the following properties:

- The center point of the object is inset is 2 inches (144 points) on the horizontal axis and 4 inches (288 points) on the vertical axis.
- The length of the radius from the center point to each corner is 1 inch.
- The polygon has 7 sides.

```
tell application "Adobe Illustrator"
set docRef to make new document
set pathRef to make new polygon in docRef with properties ¬
   {center point:{144, 288},sides:7,radius:72.0
end tell
```

6 Scripting with VBScript

This chapter uses script examples and explanations to help you to become familiar with Illustrator scripting using VBScript.

For more information

Several extended sample scripts are available in the `/Scripting/Sample Scripts` folder in your Illustrator CS3 installation directory.

For information about individual classes, objects, properties, methods and parameters, as well as script samples that demonstrate how to use many of them, refer to the *Adobe Illustrator CS3 VBScript Reference*, which is found in the `/Scripting/Documentation` folder in your Illustrator CS3 installation directory. You can also use view Illustrator CS3 type library from most VBScript editors or any Microsoft Office application. For information, see "Viewing the VBScript object model" on page 9.

If you are a beginner and find that you don't understand the concepts and terms used in this chapter, read the *Adobe Introduction to Scripting*.

Your first Illustrator script

The traditional first project in any programming language is to display the message "Hello World!"

1. Start any text editor (Notepad, for example).
2. Type the following code.

```
Rem Hello World
Set appRef = CreateObject("Illustrator.Application")
Rem Create a new document and assign it to a variable
Set documentRef = appRef.Documents.Add
Rem Create a new text frame item and assign it to a variable
Set sampleText = documentRef.TextFrames.Add
Rem Set the contents and position of the TextFrame
sampleText.Position = Array(200, 200)
sampleText.Contents = "Hello World!"
```

3. Save the file as text only in a folder of your choice. Give the file the file extension `.vbs`.
4. To test the script, do either of the following:
   - Double-click the file.
   - Start Illustrator and choose File > Scripts > Other Scripts, and then navigate to and run your script file.

Tip: To add the script to the Illustrator Scripts menu (File > Scripts), save the script in the Scripts folder. The script will appear on the menu the next time you start Illustrator. For information, see "Installing scripts in the Scripts menu" on page 10. Please note in general that, when you launch a VBScript script from the Scripts menu, any `msgBox` dialogs will not display correctly.

Adding features to "Hello World"

Next, let's create a new script that makes changes to the Illustrator document you created with your first script. The second script demonstrates how to:

- Get the active document.
- Get the width of the active document.
- Resize the text frame item to match the document's width.

Note: If you closed the Illustrator document without saving it, run your first script again to create a new document.

1. Copy the following script into your text editor, and save the file.

```
Set appRef = CreateObject("Illustrator.Application")
'Get the active document
Set documentRef = appRef.ActiveDocument
Set sampleText = documentRef.TextFrames(1)
' Resize the TextFrame item to match the document width
sampleText.Width = documentRef.Width
sampleText.Left = 0
```

2. Run the script.

Accessing and referencing objects

When you write a script, you must first decide which file, or `Document`, the script should act on. Through the `Application` object, the script can create a new document, open an existing document, or act on a document that is already open.

The script can create new objects in the document, operate on objects that the user has selected, or operate on objects in one of the object collections. The following sections illustrate various techniques for accessing, referencing, and manipulating Illustrator objects.

Obtaining objects from collections

In general, to obtain a reference to a specific object, you can navigate the containment hierarchy. For example, to use the variable `myPath` to store a reference to the first `PathItem` in the second layer of the active document:

```
Set myPath = appRef.ActiveDocument.Layers(2).PathItems(1)
```

The following scripts demonstrate how to reference an object as part of a document.

```
Set documentRef = appRef.ActiveDocument
Set pageItemRef = documentRef.PageItems(1)
```

In the script below, the variable `pageItemRef` will not necessarily refer to the same object as the above script since this script includes a reference to a layer:

```
Set documentRef = appRef.ActiveDocument
Set pageItemRef = documentRef.Layers(1).PageItems(1)
```

Note: VBScript indexes start at 1 for object collections. However, VBScript allows you to specify whether array indexes start at 1 or 0. For information on specifying the index start number for arrays, refer to any VBScript textbook or tutorial.

Creating new objects

You can use a script to create new objects. To create objects that are available from collection objects, you use the collection object's `Add` method.

```
Set myDoc = appRef.Documents.Add()
Set myLayer = myDoc.Layers.Add()
```

Some collection objects do not have an `Add` method. You create an object of this type by defining a variable and using the `CreateObject` method. For example, to create a new `CMYKColor` object using the variable name `newColor`:

```
Set newColor = CreateObject ("Illustrator.CMYKColor")
```

Working with selections

When the user makes a selection in a document, the selected objects are stored in the document's *selection* property. To access all selected objects in the active document:

```
Set appRef = CreateObject ("Illustrator.Application")
Set documentRef = appRef.ActiveDocument
selectedObjects = documentRef.Selection
```

Depending on what is selected, the *selection* property value can be an array of any type of art objects. To get or manipulate the properties of the selected art items, you must retrieve the individual items in the array. To find out an object's type, use the *typename* property.

The following sample gets the first object in the array, and then displays the object's type:

```
Set appRef = CreateObject ("Illustrator.Application")
Set documentRef = appRef.ActiveDocument
selectedObjects = documentRef.Selection
Set topObject = selectedObjects(1)
MsgBox(topObject.Typename)
```

Note: The `MsgBox` method does not display a dialog when the script is run from the Illustrator Scripts menu (File>Scripts).

Note: The first object in a selection array is the selected object that was last added to the page, and not the last object selected.

Selecting artwork objects

To select an artwork object, use the object's `Selected` property.

Working with text frames

To create a text frame of a specific type in VBScript, you use the `TextFrames` method that corresponds to the type of frame you want to create. For example:

```
Set rectRef = docRef.PathItems.Rectangle(700, 50, 100, 100)

' Use the AreaText method to create the text frame
Set areaTextRef = docRef.TextFrames.AreaText(rectRef)
```

Threaded frames

As in the Illustrator application, you can thread area or path text frames.

To thread existing text frames, you use the `NextFrame` or `PreviousFrame` property of the `TextFrames` object.

Note: When copying the following script to a script or text editor, place the value of the *Contents* property a single line. The long line continuation character (_) is not valid when enclosed in a string.

```
Set appRef = CreateObject("Illustrator.Application")
Set myDoc = appRef.Documents.Add
Set myPathItem1 = myDoc.PathItems.Rectangle(244, 64, 82, 76)
Set myTextFrame1 = myDoc.TextFrames.AreaText(myPathItem1)
  myTextFrame1.Contents = "This is two text frames linked together as one
story, with text flowing from the first to the last."
Set myPathItem2 = myDoc.PathItems.Rectangle(144, 144, 42, 116)
Set myTextFrame2 = myDoc.TextFrames.AreaText(myPathItem2)

'Use the NextFrame property to thread the frames
myTextFrame1.NextFrame = myTextFrame2

appRef.Redraw()
```

Threaded frames make a single story object

Threaded frames make a single `story` object.

To observe this, run the following JavaScript after running the script in "Threaded frames" on page 50.

```
var myDoc = app.activeDocument
alert("There are " + myDoc.textFrames.length + " text frames.")
alert("There are " + myDoc.stories.length + " stories.")
```

Creating paths and shapes

This section explains how to create items that contain paths.

Paths

To create a freeform path, you specify a series of path points, either as series of x-y coordinates or as `PathPoint` objects.

- Using x-y coordinates limits the path to straight segments only.

- To created a curved path, you must create `PathPoint` objects.

Your path can consist of a combination of page coordinates and `PathPoint` objects.

Specifying a series of x-y coordinates

To specify a path using page coordinate pairs, you use the `SetEntirePath()` method of the `PathItems` object. The following script specifies three pair of x-y coordinates to create a path that has three points.

```
Set appRef = CreateObject ("Illustrator.Application")

Set firstPath = appRef.ActiveDocument.PathItems.Add
   firstPath.Stroked = True
firstPath.SetEntirePath(Array(Array(220, 475),Array(375, 300),_
     Array(200, 300)))
```

Using path point objects

To create a `PathPoint` object, you must define three values for the point:

- A fixed *anchor* point, which is the point on the path
- A pair of direction points: the *left direction* point and the *right direction* point, which allow you to control the path segment's curve

  You define each property as an array of page coordinates in the format `(Array (x,y))`.

  - If all three properties of a `PathPoint` object have the same coordinates, and the properties of the next `PathPoint` in the line are equal to each other, you create a straight line segment.
  - If two or more properties in a `PathPoint` object hold different values, the segment connected to the point is curved.

To create a path, or to add points to an existing path, using `PathPoint` objects, you create a `PathItem` object and then add the path points as child objects in the `PathItem`.

```
Set appRef = CreateObject ("Illustrator.Application")

Set firstPath = appRef.ActiveDocument.PathItems.Add
   firstPath.Stroked = true
Set newPoint = firstPath.PathPoints.Add
'Using identical coordinates creates a straight segment
newPoint.Anchor = Array(75, 300)
newPoint.LeftDirection = Array(75, 300)
newPoint.RightDirection = Array(75, 300)

Set newPoint2 = firstPath.PathPoints.Add
newPoint2.Anchor = Array(175, 250)
newPoint2.LeftDirection = Array(175, 250)
newPoint2.RightDirection = Array(175, 250)

Set newPoint3 = firstPath.PathPoints.Add
'Using different coordinates creates a curve
newPoint3.Anchor = Array(275, 290)
newPoint3.LeftDirection = Array(135, 150)
newPoint3.RightDirection = Array(155, 150)
```

Combining path point types

The following script sample creates a path with three points:

```
Set appRef = CreateObject("Illustrator.Application")
Set myDoc = appRef.ActiveDocument
Set myLine = myDoc.PathItems.Add
   myLine.Stroked = True
   myLine.SetEntirePath( Array( Array(320, 475), Array(375, 300)));

// Append another point to the line
Set newPoint = myDoc.myLine.PathPoints.Add
   'Using identical coordinates creates a straight segment
   newPoint.Anchor = Array(220, 300)
   newPoint.LeftDirection = Array(220, 300)
   newPoint.RightDirection = Array(220, 300)
```

Shapes

To create a shape, you use the `PathItems` method that corresponds to the shape's name (such as *ellipse*, *rectangle*, or *polygon*), and use parameters to specify shape's position, size, and other characteristics such as the number of sides in a polygon.

Tip: Remember:

- All measurements and page coordinates are processed as points by the scripting engine. For more information, see "Measurement units" on page 27.
- x and y coordinates are measured from the bottom left corner of the document, as indicated in the Info palette in the Illustrator application. For information, see "Page item positioning and dimensions" on page 27.

Creating a rectangle

The following sample creates a rectangle with the following properties:

- The top of the rectangle is 2 inches (144 points) from the bottom edge of the page.
- The left edge is 2 inches (144 points) from the left edge of the page.
- The rectangle is 1 inch wide and 2 inches long.

```
Set appRef = CreateObject("Illustrator.Application")
Set frontDocument = appRef.ActiveDocument
' Create a new rectangle with
' top = 400, left side = 50, width = 150, height = 100
Set newRectangle = frontDocument.PathItems.Rectangle(400,50,150,100)
```

Creating a polygon

The following sample creates a polygon with the following properties:

- The center point of the object is inset 2 inches (144 points) on the horizontal axis and 4 inches (288 points) on the vertical axis.
- The length of the radius from the center point to each corner is 1 inch.

- The polygon has 7 sides.

```
Set appRef = CreateObject("Illustrator.Application")
Set frontDocument = appRef.ActiveDocument
' Create a new rectangle with
' top = 400, left side = 50, width = 150, height = 100
Set newPolygon = frontDocument.PathItems.Polygon(144,288,72,7)
```

Working with enumeration values

Properties that use enumeration values in VBScript use a numeral rather than a text value. For example, the `Orientation` property of the `TextFrame` object specifies whether text content is horizontal or vertical in the text frame. The property uses the `aiTextOrientation` enumeration, which has two possible values:

- `aiHorizontal`
- `aiVertical`

To find the numeral values of enumerations, use either of the following:

- The object browser in your scripting editor environment. For information, see "Viewing the VBScript object model" on page 9.
- The *Adobe Illustrator CS3 VBScript Reference*, which lists the numeral values directly after the constant value in the **Enumerations** chapter at the end of the book. The following example is taken from that table:

Enumeration Type	Values	What it means
AiTextOrientation	aiHorizontal = 0 aiVertical = 1	The orientation of text in a text frame

The following sample specifies vertical text orientation:

```
Set appRef = CreateObject ("Illustrator.Application")
Set docRef = appRef.Documents.Add
Set textRef = docRef.TextFrames.Add
textRef.Contents = "This is some text content."
textRef.Left = 50
textRef.Top = 700
textRef.Orientation = 1
```

Note: Generally, it is considered good scripting practice to place the text value in a comment following the numeral value, as in the following sample statement:

```
textRef.Orientation = 1 ' aiVertical
```

Index