Contents

1 Introduction ................................................................. 6
   What is scripting? .......................................................... 6
   Why use scripting? ......................................................... 6
   What about actions? ....................................................... 6
   Script support in Adobe Illustrator CS4 .................................. 7
      ExtendScript features .................................................. 7
      ExtendScript tools ..................................................... 7
   Script file extensions ................................................... 7
   Viewing sample scripts ................................................. 8
   Viewing the object model .............................................. 8
      Viewing the JavaScript object model ............................... 8
      Viewing the AppleScript object model ............................. 8
      Viewing the VBScript object model ................................ 9
   Executing scripts ....................................................... 9
      Installing scripts in the Scripts menu .............................. 10
      Executing scripts from the Other Scripts menu item .............. 10
      Startup scripts (.jsx scripts only) ................................ 10
   Changes since CS3 ..................................................... 11
   Known issues ............................................................ 16

2 The Illustrator Scripting Object Model .................................... 17
   Object-naming conventions ............................................ 18
   Top-level (containing) objects ....................................... 18
      Application ................................................................... 18
      Document ................................................................... 19
      Layer ....................................................................... 19
   The artwork tree .......................................................... 19
      Art styles ................................................................... 20
      Color objects ............................................................ 21
   Text objects ................................................................. 21
      Text frames .................................................................. 21
      Objects that represent text content .................................. 22
      Text styles .................................................................. 23
   Dynamic objects ........................................................... 24
   Symbols ......................................................................... 24
   Transformations ................................................................ 24
3 Scripting Illustrator ........................................... 25
   Launching and quitting Illustrator from a script .................. 25
   Launching and activating Illustrator ................................ 25
   Quitting Illustrator ............................................. 26
   Working with objects ............................................ 26
   Getting the frontmost document or layer ......................... 26
   Creating new objects ............................................ 26
   Collection objects ............................................... 27
   Selected objects ................................................ 28
   Notes on renaming objects stored in the application’s panels .......... 28
   Measurement units ............................................... 29
   Em space units ................................................ 29
   Page-item positioning and dimensions ............................. 29
   Art item bounds ............................................... 30
   Paths and shapes ............................................... 31
   User-interaction levels ......................................... 31
   Printing Illustrator documents ................................... 32

4 Scripting with AppleScript ........................................ 33
   For more information .............................................. 33
   Your first Illustrator script ....................................... 33
      Adding features to “Hello World” ............................... 34
   Object references ............................................... 34
      Obtaining objects from documents and layers .................. 35
      Creating new objects ......................................... 35
      Working with selections ..................................... 35
   Working with text frames ....................................... 36
      Threaded frames .............................................. 36
   Creating paths and shapes ..................................... 37
      Paths ........................................................ 37
      Shapes ....................................................... 38

5 Scripting with JavaScript .......................................... 40
   For more information .............................................. 40
   Your first Illustrator script ....................................... 40
      Adding features to “Hello World” ............................... 41
   Working with methods in JavaScript ........................... 41
   Accessing and referencing objects ................................ 42
      Referencing the application object ............................. 42
      Accessing objects in collections ............................... 42
      Creating new objects ......................................... 43
      Working with selections ..................................... 44
   Working with text frames ....................................... 44
      Threaded frames .............................................. 44
## Contents

Creating paths and shapes ................................................................. 45  
Paths .................................................................................................... 45  
Shapes ................................................................................................. 46  

6 **Scripting with VBScript** ............................................................ 48  
For more information ........................................................................ 48  
Your first Illustrator script ............................................................... 48  
Adding features to "Hello World" .................................................... 49  
Accessing and referencing objects ................................................... 49  
  Obtaining objects from collections ................................................. 49  
  Creating new objects ...................................................................... 50  
  Working with selections .................................................................... 50  
Working with text frames ................................................................. 51  
Threaded frames ................................................................................ 51  
Creating paths and shapes ............................................................... 51  
  Paths ................................................................................................. 51  
  Shapes ............................................................................................... 53  
Working with enumeration values .................................................... 54

**Index** ............................................................................................. 55
1 Introduction

This guide describes the scripting interface to Adobe Illustrator CS4.

If you are new to scripting or want 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, affecting only one object in the current document, or complex, affecting objects in all your Illustrator documents. The tasks might even involve other applications, like word processors, spreadsheets, and database management programs.

For the most part, the building blocks of scripting correspond to the Illustrator tools, menus, panels, and dialog boxes with which you are already an expert. If you know what you want 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 work are anything but creative. In fact, you 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 reduces the time you have available for doing creative work.

With a small investment of time and effort, 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.

Scripting also can 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 also could write a script that accesses built-in transformation matrix functions to stretch, scale, and distort a series of objects. Without scripting, you would likely miss out on the creative potential of such labor-intensive techniques.

What about actions?

Both actions and scripts are 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 scripts 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, like changing the stroke type of rectangles but not ellipses. Scripts can get information and make 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 CS4

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

**NOTE:** Also, 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 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.

For information on accessing the ESTK and the Object Model Viewer, see “Viewing the JavaScript object model” on page 8.

**ExtendScript tools**

ExtendScript also provides various tools and utilities, including the following:

- 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 other features, see *JavaScript Tools Guide*.

**Script file extensions**

For a file to be recognized by Adobe Illustrator CS4 as a valid script file, the file must have the correct file name extension:

<table>
<thead>
<tr>
<th>Script type</th>
<th>File type</th>
<th>Extension</th>
<th>Platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppleScript</td>
<td>compiled script</td>
<td><code>.scpt</code></td>
<td>Mac OS</td>
</tr>
<tr>
<td></td>
<td>OSAS file</td>
<td>(none)</td>
<td></td>
</tr>
<tr>
<td>JavaScript</td>
<td>text</td>
<td><code>.js</code></td>
<td>Windows</td>
</tr>
<tr>
<td>ExtendScript</td>
<td>text</td>
<td><code>.jsx</code></td>
<td>Mac OS</td>
</tr>
<tr>
<td>VBScript</td>
<td>text</td>
<td><code>.vbs</code></td>
<td>Windows</td>
</tr>
</tbody>
</table>
Viewing sample scripts

Adobe provides sample scripts for many objects, properties, and methods in the Illustrator CS4 DOM. You can view script samples in two locations:

➤ In the /Scripting/Sample Scripts folder in your Illustrator CS4 installation directory

➤ In the Adobe Illustrator CS4 scripting reference for your scripting language, located in the /Scripting/Documentation folder in your Illustrator CS4 installation directory

Viewing the object model

Each of the supported scripting languages provides 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, follow these steps:

1. Start the ESTK.

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

   ➤ Windows:
   `system drive\Program Files\Adobe\Adobe Utilities\ExtendScript Toolkit CS4`

   ➤ Mac OS:
   `system drive:Applications:Utilities:Adobe Utilities:ExtendScript Toolkit CS4`

2. In the ESTK, choose Help > Object Model Viewer.

3. In the Object Model Viewer window, select Adobe Illustrator CS4 Type Library from the Browser drop-down list.

Several extended sample scripts are available in the /Scripting/Sample Scripts folder in your Illustrator CS4 installation directory.

You also can view script samples and information about individual classes, objects, properties, methods, and parameters in Adobe Illustrator CS4 Scripting Reference: JavaScript, located in the /Scripting/Documentation folder in your Illustrator CS4 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.

For details of how to use 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.

3. In the Open Dictionary dialog, find and select Adobe Illustrator CS4, and click Open.

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

Several extended sample scripts are in the :Scripting:Sample Scripts folder in your Illustrator CS4 installation directory.

You also can view script samples and information about individual classes, objects, properties, methods and parameters in Adobe Illustrator CS4 Scripting Reference: AppleScript, located in the :Scripting:Documentation folder in your Illustrator CS4 installation directory.

**Viewing the VBScript object model**

VBScript provides a type library 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 probably provides access to the library. For information see your editor’s Help.

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 CS4 Type Library, and click OK.

4. Choose View > Object Browser, to display the Object Browser window.

5. Choose “Illustrator” from the list of open libraries in the top-left pull-down menu of the Object Browser window.

Several extended sample scripts are in the /Scripting/Sample Scripts folder in your Illustrator CS4 installation directory.

You also can view script samples and information about individual classes, objects, properties, methods, and parameters in Adobe Illustrator CS4 Scripting Reference: VBScript, located in the /Scripting/Documentation folder in your Illustrator CS4 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 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 also can 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, located in the /Illustrator CS4/Presets folder in your Illustrator CS4 installation directory. The script’s filename, minus the file extension, appears in the Scripts menu.

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 many 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.

Only files that are of one of the supported file types are displayed in the browse dialog. For details, see “Script support in Adobe Illustrator CS4” on page 7.

Startup scripts (.jsx scripts only)

JavaScript scripts with a .jsx file extension can be installed in one of two folders, so 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 CS4
➤ A general startup scripts folder, which contains scripts that run automatically when you start any Creative Suite 4 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 CS4 is installed to its default location, you would create the Startup Scripts folder at the following location:

➤ Windows: C:\Program Files\Adobe\Adobe Illustrator CS4\Startup Scripts\
➤ Mac OS: /Applications/Adobe Illustrator CS4/Startup Scripts/

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).
CHAPTER 1: Introduction

General startup scripts folder

The general startup scripts folder contains scripts that run automatically when you start any Creative Suite 4 application. You create the folder in the following location:

➤ Windows: Program Files/Common Files/Adobe/Startup Scripts CS4/Illustrator
➤ Mac OS: :Library:Application Support:Adobe:Startup Scripts CS4:Illustrator

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 like the following:

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

For details, see JavaScript Tools Guide.

Changes since CS3

This section lists changes made to the scripting object model to support features in Illustrator CS4.

➤ A new type, DocumentArtboardLayout, sets the layout of artboards in a document. It is used by the DocumentPreset class and the Add method in the Document class. Possible layouts are GridByRow, GridByCol, Row, Column, RLGridByRow, RLGridByCol, and RLRow; for example:

  ➤ AppleScript — artboardLayout: grid by row
  ➤ JavaScript — DocumentArtboardLayout.GridByRow
  ➤ VBScript — AiDocumentArtboardLayout.aiGridByRow

➤ Five new properties in the Add method of the Documents collection — NumArtboards, ArtboardLayout, ArtboardSpacing, ArtboardRowsOrCols, and MaxCanvasSizeEnabled — support the new, multiple-artboard feature.

➤ Five new properties in the DocumentPreset class — NumArtboards, ArtboardLayout, ArtboardSpacing, ArtboardRowsOrCols, and MaxCanvasSizeEnabled — support the new multiple-artboard feature.

➤ Two new properties — ArtboardRange and SaveMultipleArtboards — support the new multiple-artboards feature. They apply to several classes: ExportOptionsFlash, ExportOptionsPhotoshop, EPSSaveOptions, MXGSaveOptions, and PDFSaveOptions (ArtboardRange only).

➤ A new class, NonNativeItem, represents a non-native item in an Illustrator document. This is another subclass of PageItem. Its parent can be a Document, Group, or Layer.

  ➤ AppleScript — non native item
  ➤ JavaScript — nonNativeItem
  ➤ VBScript — NonNativeItem
A new collection, NonNativeItems, contains a collection of NonNativeItem objects.

- **AppleScript** — non native items of document
- **JavaScript** — app.activeDocument.nonNativeItems
- **VBScript** — App.ActiveDocument.NonNativeItems

A new FXG file format supports designer-developer workflow, by allowing an Illustrator document to be saved as an FXG file containing XML-style descriptions of Illustrator graphical items, which can then be used in other FXG/XML-based applications.

- **AppleScript** — fxg
- **JavaScript** — DocumentType.FXG
- **VBScript** — AiDocumentType.aiFXG

Two new PDFX standards were added, PDFX32002 and PDFX42007.

A new type, FXGVersion, for setting the FXG file format version. There is only one option, FXG version 1.0.

- **AppleScript** — version: version 1.0
- **JavaScript** — FXGVersion.VERSION1.0
- **VBScript** — AiFXGVersion.aiVersion1.0

A new type, FiltersPreservePolicy, which defines whether to preserve appearance and editability of filters when saving to FXG. Available options are ExpandFilters, RasterizeFilters, and KeepFiltersEditable.

- **AppleScript** — filters policy: keep filters editable
- **JavaScript** — FiltersPreservePolicy.KEEPFILTERSEDITABLE
- **VBScript** — AiFiltersPreservePolicy.aiKeepFiltersEditable

A new type, GradientsPreservePolicy, which defines whether to preserve appearance and editability of gradients when saving to FXG. Available options are RasterizeGradients and KeepGradientsEditable.

- **AppleScript** — gradients policy: keep gradients editable
- **JavaScript** — GradientsPreservePolicy.KEEPGRADIENTSEDITABLE
- **VBScript** — AiGradientsPreservePolicy.aiKeepGradientsEditable

A new type, TextPreservePolicy, which defines whether to preserve appearance and editability of text when saving to FXG. Available options are OutlineText, RasterizeText, and KeepTextEditable.

- **AppleScript** — text policy: keep text editable
- **JavaScript** — TextPreservePolicy.KEEPTEXTEDITABLE
- **VBScript** — AiTextPreservePolicy.aiKeepTextEditable
> A new `AutoKernType.MetricsRomanOnly`, specifies metrics-Roman-only kerning.

  ➢ AppleScript — character: MetricsRomanOnly, character style: MetricsRomanOnly, line: MetricsRomanOnly, paragraph: MetricsRomanOnly, paragraph style: MetricsRomanOnly, word: MetricsRomanOnly

  ➢ JavaScript — `AutoKernType.METRICSROMANONLY`

  ➢ VBScript — `AiAutoKernType.aiMetricsRomanOnly`

> A new `FlashExportStyle.ArtBoardsToFiles`, enables each artboard in a document to be exported to a separate SWF file.

  ➢ AppleScript — export style: Artboards to Files

  ➢ JavaScript — `FlashExportStyle.ARTBOARDSTOFILES`

  ➢ VBScript — `AiFlashExportStyle.aiArtboardsToFiles`

> The `CropBounds PrintingBounds` constant has become obsolete, with the modification of artboards to support the new, multiple-artboard feature.

> Support for JIS90 and JIS04 glyph encoding/forms was added to `AlternateGlyphsForm`; for example:

  ➢ AppleScript — alternate glyphs: jis90

  ➢ JavaScript — `AlternateGlyphsForm.JIS90`

  ➢ VBScript — `AiAlternateGlyphsForm.aiJIS90Form`

> New properties were added to the `Application` class: build number and locale; for example:

  ➢ AppleScript — build number of application

  ➢ JavaScript — `app.buildNumber`

  ➢ VBScript — `App.BuildNumber`

> A new property in the `Document` class, `RasterEffectSettings`, contains a document’s raster effect settings.

  ➢ AppleScript — raster effect settings of document 1

  ➢ JavaScript — `app.activeDocument.rasterEffectSettings`

  ➢ VBScript — `App.ActiveDocument.RasterEffectSettings`

> A new method was added to the `Application` class, `ConvertSampleColors`, which converts a color from one color space to another.

  ➢ AppleScript — `Set cmykColor to convert sample color source color space RGB source color rgbColor destination color space CMYK color conversion purpose default purpose`

  ➢ JavaScript — `var cmykColor = app.convertSampleColor(ImageColorSpace.RGB, newArray(93, 32, 10), ImageColorSpace.CMYK, ColorConvertPurpose.defaultpurpose);`

  ➢ VBScript — `cmykColor = appRef.ConvertSampleColor(2, Array(93, 32, 10), 3, 0)`
Two new methods were added to the Document class. WindowCapture captures the current document to a TIFF image. Rasterize rasterizes the source art within a specified clip bounds.

- **AppleScript** — capture document 1, rasterize document 1
- **JavaScript** — `app.activeDocument.windowCapture, app.activeDocument.rasterize`

A new method in the Swatches collection, GetSelected, returns a SwatchList of Swatch objects currently selected.

- **AppleScript** — get selected swatches of document 1
- **JavaScript** — `app.activeDocument.swatches.getSelected`

A new class, SwatchGroup, contains a group of Swatch objects. It has two methods: GetAllSwatches and AddSwatch.

- **AppleScript** — swatch group 1 of document 1
- **JavaScript** — `app.activeDocument.swatchGroups[0]`
- **VBScript** — `App.ActiveDocument.SwatchGroups(1)`

A new collection, SwatchGroups, contains all SwatchGroup objects in a document. It has two methods: Remove and Add.

- **AppleScript** — swatch groups in document 1
- **JavaScript** — `app.activeDocument.swatchGroups`
- **VBScript** — `App.ActiveDocument.SwatchGroups`

A new type, SwatchList, defines a collection of Swatch objects. It is used by the SwatchGroup object and Swatches collection.

A new property in the GradientStop class, opacity, allows each stop to define its own degree of transparency. Alpha blending occurs between stops.

- **AppleScript** — stop opacity of gradient stop
- **JavaScript** — `gradientStop.opacity`
- **VBScript** — `GradientStop.Opacity`

A new property in the OpenOptionsAutoCAD class, MergeLayers, defines whether the layers in the AutoCAD file being opened should be merged into a single layer in the Illustrator document.

- **AppleScript** — merge layers: true
- **JavaScript** — `mergeLayers = true`
- **VBScript** — `MergeLayers = true`
A new class, `Artboard`, defines an artboard object or artboard in a CS4 document. It has the following properties: `ArtboardRect`, `RulerPAR`, `ShowRulers`, `ShowCenter`, `ShowCrossHairs`, `ShowSafeAreas`, and `ShowScreenEdge`.

- **JavaScript** — `app.activeDocument.artboards[0]`
- **AppleScript** — `artboard 1 of document 1`
- **VBScript** — `App.ActiveDocument.Artboards(1)`

A new collection, `Artboards`, contains all `Artboard` objects in a document. It has the following methods: `Remove`, `Add`, `GetActiveArtboardIndex`, and `SetActiveArtboardIndex`.

- **AppleScript** — `artboards of document 1`
- **JavaScript** — `app.activeDocument.artboards`
- **VBScript** — `App.ActiveDocument.Artboards`

Two new properties in the `PrintJobOptions` class, `PrintAllArtboards` and `ArtboardRange`, support printing of multiple artboards.

- **JavaScript** — `PrintJobOptions/printAllArtboards `
- **AppleScript** — `color model: grayscale rasterization`
- **VBScript** — `AiRasterizationColorModel.aiGrayscale`

A new type, `RasterizationColorModel`, enables the color model to be specified as default, grayscale, or bitmap; for example:

- **AppleScript** — `color model: grayscale rasterization`
- **JavaScript** — `RasterizationColorModel.GRAYSCALE`
- **VBScript** — `AiRasterizationColorModel.aiGrayscale`

A new type, `AntiAliasingMethod`, controls the type of rasterization method used in the rasterization: none, art optimized, and type optimized; for example:

- **AppleScript** — `antialiasing method: art optimized`
- **JavaScript** — `AntiAliasingMethod.ARTOPTIMIZED`
- **VBScript** — `AiAntiAliasingMethod.aiArtOptimized`

A new class, `RasterEffectOptions`, specifies a document’s raster-effect settings. Its properties are `ColorModel`, `Resolution`, `Transparency`, `AntiAliasing`, `ClippingMask`, `ConvertSpotColors`, and `Padding`.

- **AppleScript** — `raster effect settings of document 1`
- **JavaScript** — `app.activeDocument.rasterEffectSettings`
- **VBScript** — `App.ActiveDocument.RasterEffectSettings`

A new class, `RasterizeOptions`, specifies options that may be used when rasterizing artwork. Its properties are `ColorModel`, `Resolution`, `Transparency`, `AntiAliasing`, `ClippingMask`, `ConvertSpotColors`, `ConvertTextToOutlines`, `IncludeLayers`, `BackgroundBlack`, and `Padding`.

- **AppleScript** — `rasterize document 1 uses rasterize options`
- **JavaScript** — `app.activeDocument.rasterize uses rasterizeOptions`
- **VBScript** — `App.ActiveDocument.Rasterize uses RasterizeOptions`
Known issues

➤ Scripts that create, save, and close many Illustrator files should periodically quit and relaunch Illustrator. The recommended maximum number of files to process before quitting and relaunching Illustrator is:

- Windows 500 files
- Mac OS 1000 files

For more information on quitting and relaunching Illustrator, see “Launching and activating Illustrator” on page 25 and “Quitting Illustrator” on page 26.

➤ The “An Illustrator error occurred: 1346458189 ("PARM")” alert may be popped when badly written scripts are repeatedly run in Illustrator from the ESTK.

Scripters need to be very careful about variable initialization and namespace conflict when pushing a batch of Illustrator scripts repeatedly for execution in Illustrator via the ESTK in one 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 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 different scripts are clobbering each other.
A good understanding of the Illustrator object model will improve your scripting abilities. The following figure 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.

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 JavaScript Tools Guide.
Object-naming conventions

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

- **AppleScript names** are 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.

Throughout this document, names of properties, methods, and object are in a monospaced 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** like installed fonts (the `text fonts` property) and printers (the `printer list` property).

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

- **Application information** like the installation path, version, and whether Illustrator is visible.

- **The current active document**; that is, the art canvas that is displayed and accepting user input.

- **All open documents**.

The `application` object's 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’s properties give you access to the document’s content; for example:

- The current selection, or art objects that the user selected in the document
- All contained art objects, called page items, that make up the artwork tree
- Art objects of particular types, like symbols and text frames
- All 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 was saved since the last alteration of content
- The path of the associated file

The document object’s 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 they can 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’s opacity (overall transparency) and z order position (position in the stacking order).
- Art-creation preferences for the layer, like 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:

- compound path item
- graph item
- group item
- legacy text item
The artwork tree

➤ mesh item
➤ non native item
➤ path item
➤ placed item
➤ plugin item
➤ raster item
➤ symbol item (see “Dynamic objects” on page 24)
➤ text frame

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.

Also, 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:

AppleScript    set myPathItem to make new path item in current document
JavaScript     var myPathItem = activeDocument.pathItems.add();
VBScript        Set myPathItem = appRef.ActiveDocument.PathItems.Add()

The following artwork collections do not allow the creation of new objects using the make command or add method:

➤ graph items object
➤ mesh items object
➤ plugin items object
➤ legacy text items object

For details on creating objects of these types, see the Adobe Illustrator CS4 Scripting Reference for your language.

Art styles

Your script can apply a graphic style to artwork using the graphic style object. To apply a graphic style, 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 also can 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.

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 more text. Finally, create a rectangle and type some text inside it. Now run the following JavaScript script:

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

Text frames

There are three types of text frames:

➤ point

➤ area

➤ path
To create a specific kind of text frame, use the `kind` property of the `text frames` object in AppleScript. 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, 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 in this manual for your scripting language.

**Text geometry**

While the three kinds of text frames have common characteristics, like `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 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 the characters that fit on one 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 also are 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, like font, capitalization, and 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. They control character features like font, alignment, leading, language, and capitalization, which are properties of the character attribute object.

- paragraph style objects apply to paragraphs. They control paragraph features like first line indent, left indent, and right indent, which are properties of the paragraph attribute object.
Dynamic objects

By creating dynamic objects, you can create data-driven graphics. In the Illustrator application, you use the Variables panel to create or edit variables like graph data, linked file, text string, and visibility, 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 details on Illustrator variables, dynamic objects, and data-driven graphics, see Illustrator Help.

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

Symbols

In Illustrator, symbols are art items that are stored in the Symbols panel. Your scripts can create, delete, and duplicate `symbol` objects. When you create `symbol` objects in your script, Illustrator adds them to the Symbols panel 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 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; therefore, they 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. There are advantages to using matrices:

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

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

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 is an overview of how to use scripting objects to program Illustrator CS4. Specific examples for the supported scripting languages are in succeeding chapters.

Launching and quitting Illustrator from a script

Your scripts can control the activation and termination of Illustrator.

Launching and activating Illustrator

AppleScript

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
```

JavaScript

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

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

VBScript

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. If Illustrator is launched as an invisible application you must manually activate the application to make it visible:

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

  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 a 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"
  For Illustrator CS4, use "Illustrator.Application.CS4"
➤ Use the New operator if you 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

#### AppleScript

Use the quit command:

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

#### JavaScript

Use the `app.quit()` method:

```
app.quit()
```

#### VBScript

Use the Application object’s Quit method:

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

### Working with objects

#### Getting the frontmost document or layer

To refer to the selected document, 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.

There are other types of “active” or “current” object properties, like `active dataset` or `active view`. For details, see the Adobe Illustrator CS4 Scripting Reference for your language.

#### Creating new objects

Several objects (besides the application object itself) cannot be obtained from containers or parent objects. Your script must create these objects directly.
The following objects must be created explicitly:

- CMYK color
- document preset
- EPS save options
- export options AutoCAD
- export options Flash
- export options GIF
- export options JPEG
- export options Photoshop
- export options PNG8
- export options PNG24
- export options SVG
- file
- folder
- gradient color
- gray color
- Illustrator save options
- ink
- ink info
- lab color
- matrix
- MXG save options
- no color
- open options
- open options AutoCAD
- open options FreeHand
- open options PDF
- open options Photoshop
- paper info
- Pattern color
- PDF save options
- PPD file
- PPD file info
- print color management options
- print color separation options
- print coordinate options
- printer
- printer info
- print flattener options
- print font options
- print job options
- print options
- print page marks options
- print paper options
- print postscript options
- raster effect options
- rasterize options
- screen
- screen spot function
- RGB color
- spot color
- tracing options

The 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 JavaScript Tools Guide.

For information on creating an object explicitly, see the chapter for your scripting language.

**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, refer to either containing of these objects. For example, see the language-specific sections below.

**AppleScript**

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
```

**JavaScript**

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]
```
VBScript

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, see the document object Elements table in Adobe Illustrator CS4 Scripting Reference: AppleScript or the Properties table in Adobe Illustrator CS4 Scripting Reference: JavaScript or Adobe Illustrator CS4 Scripting Reference: VBScript. A diagram of the Illustrator CS4 object model is in “The Illustrator Scripting Object Model” on page 17.

Selected objects

Sometimes, you want to write scripts that act on 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, use the select command or method of the text range object.

Selecting art items

You can select an art object (like graph items, mesh items, raster items, and symbol items) 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, use the document object’s selection property. To work with the objects in the selection array, you must determine their type, so you will know which properties and methods or commands you can use with them. In JavaScript and VBScript, each artwork object type has a read-only typename property 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 panels

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 panel. If a script modifies the name of such an object, references to that object by index can become invalid.

Brush
Gradient
Graphic Style
Pattern
Swatch
Symbol
Variable
Measurement units

Illustrator uses points as the unit of measurement for almost all distances. One inch equals 72 points. The exception is values for properties like kerning, tracking, and the aki properties (used for Japanese text composition), which use em units. (See “Em space units” on page 29.)

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 conversion formulas for various units of measurement:

<table>
<thead>
<tr>
<th>Unit</th>
<th>Conversion formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>centimeters</td>
<td>28.346 points = 1 centimeter</td>
</tr>
<tr>
<td>inches</td>
<td>72 points = 1 inch</td>
</tr>
<tr>
<td>millimeters</td>
<td>2.834645 points = 1 millimeter</td>
</tr>
<tr>
<td>picas</td>
<td>12 points = 1 pica</td>
</tr>
<tr>
<td>Qs</td>
<td>0.709 point = 1 Q (1 Q equals 0.23 millimeter)</td>
</tr>
</tbody>
</table>

JavaScript provides the UnitValue object type, which offers unit-conversion utilities. For details, see JavaScript Tools Guide.

Em space units

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

An em is 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. In a 10-point font, a kerning value of 20 em units is equivalent to:

$$(20 \text{ units} \times 10 \text{ points}) / 1000 \text{ units/em} = 0.2 \text{ points}$$

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.

For information on the types of objects that comprise the page items collection, see “The artwork tree” on page 19.

A point is designated by a pair of coordinates:

- The horizontal position, x
- The vertical position, y
You can see these coordinates in the Info panel 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.

Also, each page item object has width and height properties. The maximum value allowed for the width or height of a page item is 16348 points.

### Art item bounds

Every page item object 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 JavaScript naming conventions.
CHAPTER 3: Scripting Illustrator

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.

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 details, samples, and information on creating shapes, see the chapter for your scripting language.

User-interaction levels

When user feedback is required, an application typically presents a dialog. This is called user interaction. It 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.

AppleScript

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

<table>
<thead>
<tr>
<th>Property Value</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>never interact</td>
<td>No interaction is allowed.</td>
</tr>
<tr>
<td>interact with self</td>
<td>Interact only with scripts executed from the Scripts menu (File &gt; Scripts).</td>
</tr>
<tr>
<td>interact with local</td>
<td>Interact with scripts executed on the local machine (including self).</td>
</tr>
<tr>
<td>interact with all</td>
<td>Interact with all scripts.</td>
</tr>
</tbody>
</table>
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.

**JavaScript**

In JavaScript, there are two possible values for the `app.userInteractionLevel` property:

<table>
<thead>
<tr>
<th>Property Value</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISPLAYALERTS</td>
<td>Interaction is allowed.</td>
</tr>
<tr>
<td>DONTDISPLAYALERTS</td>
<td>No interaction is allowed.</td>
</tr>
</tbody>
</table>

**VBScript**

In VBScript, there are two possible values for the `UserInteractionLevel` property of the `Application` object:

<table>
<thead>
<tr>
<th>Property Value</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>aiDisplayAlerts</td>
<td>Interaction is allowed.</td>
</tr>
<tr>
<td>aiDontDisplayAlerts</td>
<td>No interaction is allowed.</td>
</tr>
</tbody>
</table>

**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.

Illustrator supports at most one print session at a time, because of limits in the current printing architecture.

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

The `print options` object allows you to define print settings like PPD, PostScript options, paper options, and color-management options. 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`, `PPD file list`, and `print presets list` properties.
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 in the :Scripting:Sample Scripts folder in your Illustrator CS4 installation directory.

For information about individual classes, objects, properties, commands, and parameters, as well as script samples that demonstrate how to use many of these items, see Adobe Illustrator CS4 Scripting Reference: AppleScript, in the :Scripting:Documentation folder in your Illustrator CS4 installation directory. You also can view the Illustrator CS4 dictionary from the Script Editor application; see “Viewing the AppleScript object model” on page 8.

If you do not understand the concepts and terms used in this chapter, read Adobe Introduction to Scripting.

Your first Illustrator script

The traditional first project in any programming language is displaying the message “Hello World!” In this example, you create a new Illustrator document, then add a text frame containing this message. Follow these steps:

1. Open Script Editor.
   
   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 details, see “Installing scripts in the Scripts menu” on page 10.
Adding features to “Hello World”

Next, we create a new script that makes changes to the Illustrator document you created with your first script. Our second script demonstrates how to:

➤ Get the active document.
➤ Get the width of the active document.
➤ Resize the text frame to match the document’s width.

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

Follow these steps:

1. In Script Editor, choose File > New to create a new script.
2. Enter the following code:

```applescript
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, this is a reference to the first path in layer 2:

```
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 becomes `path item 1 of layer 2 of document 1`. This new object displaces the original path item, forcing the original to index position 2; therefore, any references made to `path item 1 of layer 2 of document 1` refer to the new object. This method of applying index numbers assures that lowest index number refers to the object that was worked on most recently.

Consider the following sample script:

```applescript
-- 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 consecutive `make` command. (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. We recommend 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 `pageItemRef` variable 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, 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, 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
```

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

**Selecting artwork objects**

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

---

**Working with text frames**

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

```
set myRect to make new rectangle in current document with properties
    {position:{100, 700}, height:100, width:100}
s
set myAreaText to 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}
m
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**

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

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

When copying the following script to your script editor, place the value of the `contents` property on one 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}
m
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}
m
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
```
CHAPTER 4: Scripting with AppleScript

Creating paths and shapes

This section explains how to create items that contain paths.

**Paths**

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

Using x-y coordinates limits the path to straight segments. To created a curved path, you must create `pathpoint` objects. A path can comprise a combination of page coordinates and `pathpoint` objects.

**Specifying a series of x-y coordinates**

To specify a path using page-coordinate pairs, use the `entire path` property of the `pathitems` object. The following script specifies three pairs of x-y coordinates, to create a path with three points:

```applescript
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 `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—left direction and right direction—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 have different values, the segment connected to the point is curved.

To create a path or add points to an existing path using `pathpoint` objects, create a `pathitem` object, then add the path points as child objects in the `pathitem`:

```
```
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 (like ellipse, rectangle, or
polygon), and use the object's properties to specify the shape's position, size, and other information like
the number of sides in a polygon.

Remember:

➤ The scripting engine processes all measurements and page coordinates as points. For details, see
“Measurement units” on page 29.

➤ x and y coordinates are measured from the bottom-left corner of the document, as indicated in the
Info panel in the Illustrator application. For details, see “Page-item positioning and dimensions” on
page 29.
Write-once access

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

Creating a rectangle

Consider the following sample:

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

The sample creates a rectangle with these properties:

➤ The top-right corner of the object 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.

Creating a polygon

Consider the following sample:

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

The sample creates a polygon with these 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 polygon has 7 sides.

➤ The length of the radius from the center point to each corner is 1 inch (72 points).
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 in the `/Scripting/Sample Scripts` folder in your Illustrator CS4 installation directory.

For information about individual classes, objects, properties, methods, and parameters, as well as script samples that demonstrate how to use many of these items, see *Adobe Illustrator CS4 Scripting Reference: JavaScript*, in the `/Scripting/Documentation` folder in your Illustrator CS4 installation directory. You also can use the Illustrator dictionary, which you access from the Object Model Viewer in the ESTK. For information on using the ExtendScript Toolkit and the Object Model Viewer, see “Viewing the JavaScript object model” on page 8 or *JavaScript Tools Guide*.

If you do not understand the concepts and terms used in this chapter, read *Adobe Introduction to Scripting*.

Your first Illustrator script

The traditional first project in any programming language is displaying the message “Hello World!” In this example, you create a new Illustrator document, then add a text frame containing this message. Follow these steps:

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

   ```javascript
   //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!"
   ```

   For information on locating the ExtendScript Toolkit, see “Viewing the JavaScript object model” on page 8.

2. To test the script, do either of the following:

   ➣ If you are using the ESTK, select Adobe Illustrator CS4 from the drop-down list in the upper-left corner, select Yes to start Illustrator, then choose Debug > Run in the ESTK to run the script.

   ➣ If you are using a different text editor than the ESTK, save the file as text-only in a folder of your choice, using the file extension `.jsx`, then start Illustrator. In Illustrator, choose File > Scripts > Other Scripts, and 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 details, see “Installing scripts in the Scripts menu” on page 10.
Adding features to “Hello World”

Next, we create a new script that makes changes to the Illustrator document you created with your first script. Our second script demonstrates how to:

➤ Get the active document.
➤ Get the width of the active document.
➤ Resize the text frame to match the document’s width.

If you already closed the Illustrator document, run your first script again to create a new document, before proceeding with this exercise.

Follow these steps:

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

   ```javascript
   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 want 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.

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

In the definition, taken from *Adobe Illustrator CS4 Scripting Reference: JavaScript*, optional parameters are enclosed in square brackets ([ ]).

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

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

You need to specify undefined for the changePositions and changeFillPatterns parameters. You do not have to specify anything for the two optional parameters following changeFillGradients, since they are at the end of the parameter list.
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 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. Because all JavaScript scripts are executed from within the Illustrator application, however, 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, use the app global variable. 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 name. The collection object takes the plural form of the object name. For example, the collection object for the document object is documents.

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: the first object in the collection has the index [0].

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

To access the first style in a 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:

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

- Using the containment hierarchy:

  ```javascript
  var firstStyle = app.activeDocument.graphicStyles[0];
  ```
The following statements assign the name of the first graphic style in the collection to a variable. You can use these statements interchangeably.

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

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 is `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 ];
```

Note 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];
```

Each element in the collection is an object of the desired type, and you can access its properties through the collection. For example, to get an object's name, 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, 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. To create an object of this type, define a variable, then use 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:

```javascript
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, then displays the object's type:

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

The first object in a selection array is the selected object that was last added to the page, 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, use the `textFrames` method whose name corresponds to the text frame's type; for example:

```javascript
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 text frames or path text frames.

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

When copying the following script to the ESTK, place the value of the `contents` property on one line.

```javascript
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 44.

```javascript
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, specify a series of path points, as a series of x-y coordinates or pathPoint objects.

Using x-y coordinates limits the path to straight segments. To created a curved path, you must create pathPoint objects. Your path can comprise a combination of page coordinates and pathPoint objects.

Specifying a series of x-y coordinates

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

```javascript
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—left direction and right direction—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 add points to an existing path using pathPoint objects, create a pathItem object, 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.stroke = 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:

```javascript
var myDoc = app.activeDocument;
var myLine = myDoc.pathItems.add();
  myLine.stroke = 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, use the `pathItems` method that corresponds to the shape's name (like `ellipse`, `rectangle`, or `polygon`), and use the parameters to specify shape's position, size, and other information like the number of sides in a polygon.

Remember:

- All measurements and page coordinates are processed as points by the scripting engine. For details, see "[Measurement units](#) on page 29.

- x and y coordinates are measured from the bottom-left corner of the document, as indicated in the Info panel in the Illustrator application. For details, see "[Page-item positioning and dimensions](#) on page 29."
Creating a rectangle

Consider the following sample:

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

The sample uses the `pathItems` object's `rectangle()` method to create a rectangle with these 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 (72 points) wide and 3 inches (216 points) long.

Creating a polygon

Consider the following sample:

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

The sample uses the `polygon()` method to create a polygon with these 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 (72 points).
- The polygon has 7 sides.
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 in the /Scripting/Sample Scripts folder in your Illustrator CS4 installation directory.

For information about individual classes, objects, properties, methods, and parameters, as well as script samples that demonstrate how to use many of these items, see Adobe Illustrator CS Scripting Reference: VBScript, in the /Scripting/Documentation folder in your Illustrator CS4 installation directory. You also can view the Illustrator CS4 type library from most VBScript editors or any Microsoft Office application; see “Viewing the VBScript object model” on page 9.

If you do not understand the concepts and terms used in this chapter, read Adobe Introduction to Scripting.

Your first Illustrator script

The traditional first project in any programming language is displaying the message “Hello World!” Follow these steps:

1. Start any text editor (for example, Notepad).

2. Type the following code:

   ```vb
   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, using the file extension .vbs.

4. To test the script, do one of the following:
   - Double-click the file.
   - Start Illustrator, choose File > Scripts > Other Scripts, and 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 details, see “Installing scripts in the Scripts menu” on page 10. In general, when you launch a VBScript script from the Scripts menu, any `msgBox` dialogs will not display correctly.
Adding features to “Hello World”

Next, we 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.

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

Follow these steps:

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

   ```vbscript
   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 selected, or operate on objects in one of the object collections. The following sections illustrate techniques for accessing, referencing, and manipulating Illustrator objects.

Obtaining objects from collections

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

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

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

```vbscript
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:

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

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, see 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, use the collection object’s `Add` method:

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

Some collection objects do not have an `Add` method. To create an object of this type, define a variable and use the `CreateObject` method. For example, the following code creates a new `CMYKColor` object using the variable name `newColor`:

```vbscript
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:

```vbscript
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, then displays the object’s type:

```vbscript
Set appRef = CreateObject("Illustrator.Application")
Set documentRef = appRef.ActiveDocument
selectedObjects = documentRef.Selection
Set topObject = selectedObjects(0)
MsgBox(topObject.Typename)
```

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

The first object in a selection array is the selected object that was last *added* to the page, 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, use the `TextFrames` method that corresponds to the type of frame you want to create:

```vbnet
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 path frames or path text frames.

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

When copying the following script to a script or text editor, place the value of the `Contents` property on one line. The long-line continuation character (\_) is not valid when enclosed in a string.

```vbnet
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 VBScript after running the script in "Threaded frames" on page 51.

```vbnet
Set myDoc = appRef.ActiveDocument
myMsg = "alert(""There are " & CStr(myDoc.TextFrames.Count) & " text frames. ")"
appRef.DoJavaScript myMsg
myMsg = "alert(""There are " & CStr(myDoc.Stories.Count) & " stories. ")"
appRef.DoJavaScript myMsg
```

Creating paths and shapes

This section explains how to create items that contain paths.

Paths

To create a freeform path, specify a series of path points, as a series of either x-y coordinates or `PathPoint` objects.
Using x-y coordinates limits the path to straight segments. To created a curved path, you must create PathPoint objects. Your path can comprise a combination of page coordinates and PathPoint objects.

**Specifying a series of x-y coordinates**

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

```vbs
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—left direction and right direction—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 add points to an existing path using PathPoint objects, create a PathItem object, then add the path points as child objects in the PathItem:

```vbs
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:

```vbscript
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 = 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, use the PathItems method that corresponds to the shape's name (like ellipse, rectangle, or polygon), and use parameters to specify the shape's position, size, and other characteristics like the number of sides in a polygon.

Remember:

- The scripting engine processes all measurements and page coordinates as points. For details, see “Measurement units” on page 29.
- x and y coordinates are measured from the bottom-left corner of the document, as indicated in the Info panel in the Illustrator application. For details, see “Page-item positioning and dimensions” on page 29.

Creating a rectangle

Consider the following sample:

```vbscript
Set appRef = CreateObject("Illustrator.Application")
Set frontDocument = appRef.ActiveDocument
' Create a new rectangle with
' top = 144, left side = 144, width = 72, height = 144
Set newRectangle = frontDocument.PathItems.Rectangle(144,144,72,144)
```

The sample creates a rectangle with these 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 (72 points) wide and 2 inches (144 points) long.
Creating a polygon

Consider the following sample:

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

The sample creates a polygon with these 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 (72 points).
- The polygon has 7 sides.

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 in the text frame is horizontal or vertical. The property uses the `aiTextOrientation` enumeration, which has two possible values, `aiHorizontal` and `aiVertical`.

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

- The object browser in your scripting editor environment. See "Viewing the VBScript object model" on page 9.
- The Adobe Illustrator CS4 Scripting Reference: VBScript, which lists the numeral values directly after the constant value in the “Enumerations” chapter at the end of the book. The following example is from that table:

<table>
<thead>
<tr>
<th>Enumeration type</th>
<th>Values</th>
<th>What it means</th>
</tr>
</thead>
<tbody>
<tr>
<td>AiTextOrientation</td>
<td>aiHorizontal = 0</td>
<td>The orientation of text in a text frame</td>
</tr>
<tr>
<td></td>
<td>aiVertical = 1</td>
<td></td>
</tr>
</tbody>
</table>

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
```

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

### A

- actions, about, 6
- Adobe Illustrator
  - Plug-in Software Development Kit Function Reference, 31
- aki properties, 29
- anchor points, 31
- AppleScript
  - dictionary, 8
  - file extensions, 7
  - naming conventions, 18
  - application version, 25
  - applying styles, about, 23
  - attributes, about, 23

### C

- centimeters, conversion, 29
- character styles
  - See also fonts
  - about, 23
- clipboard, clearing before quitting, 25
- control bounds, 30
- coordinates, about, 29
- CS2 version changes, 16

### D

- datasets, using, 24
- dialogs
  - enabling, 31
  - suppressing, 31
- dimensions, page items, 29
- documents
  - page item positioning, 29
  - printing, 32

### E

- em space units, 29
- enumeration values, 54
- executing scripts, 9, 10
- ExtendScript file extension, 7

### F

- file extensions for valid scripts, 7
  - fixed points, 29
  - fixed rectangles, 30
  - fonts
    - See also character styles
    - em space units, 29
  - frames, text, 21

### G

- geometric bounds, 30

### H

- height, maximum value allowed, 29
- "Hello World" script
  - creating, 33, 40, 48
  - improving, 34, 41, 49

### I

- Illustrator
  - launching, 25
  - quitting, 25
  - specifying a version, 25
- Illustrator, See Adobe Illustrator
- inches, conversion of measurements, 29
- installing scripts, 10

### J

- JavaScript
  - changes in Illustrator CS2, 16
  - file extension, 7
  - naming conventions, 18
  - object model viewer, 8
  - .jsx extension, 7

### L

- launching Illustrator, 25
- left direction, 31
- lines, creating, 22
- local attributes, 23
matrices, about, 24
matrix class, 24
measurement values, 29
methods, using, 41
millimeters, conversion, 29

object model
changes in Illustrator CS2, 16
diagram, 17
text, 21
object references
about, 26
AppleScript, 34
objects
cannot be created by a script, 27, 28
creating in AppleScript, 35
creating in JavaScript, 26
creating in Visual Basic, 49
dimensions, 29
direct creation required, 26
hierarchy, 17
selecting, 50

page items
bounds, 30
positioning, 29
positioning and dimensions, 29
parameters, omitting, 41
paths
about, 31
creating, 51
picas, conversion, 29
points
conversion, 29
fixed, 29
zero, 29
printing
about, 21
settings options, 32

Qs (unit), conversion, 29
quitting Illustrator, 25

rectangles
creating, 53
fixed, 30
references, object. See object references
right direction, 31

script examples
creating a curved path, 37, 45, 52
creating a path, 37, 45, 52
creating a polygon, 39, 47, 54
creating a rectangle, 39, 47
creating objects, 43
selections, 50

scripting
about, 6
using, 6

scripting samples
creating a rectangle, 53
creating new objects, 50

scripts
executing, 9, 10
file extensions, 7
installing, 10
menu, 7
support in Illustrator, 7
SDK, 31
selecting objects, 50
selections
determining content, 35, 44, 50
using, 35, 44, 50

Software Development Kit, 31
stories, about, 21
symbols
about, 24
items, 24

text
art items, 21
frame types, 21
ranges. See text ranges
text ranges
content, 23
using text art, 21
transformation matrices, about, 24
Index

U
units of measurement, 29
user interaction levels, 31

V
variables
deleting, 24
using, 24
VBScript
enumeration values, 54
file extension, 7
naming conventions, 18
type library, 9
versions of Illustrator, specifying, 25
visible bounds, 30

W
width, maximum value allowed, 29
write-once, 39

X
X axis, 29

Y
Y axis, 29

Z
zero point, 29