ADOBE® ILLUSTRATOR® CC

ADOBE ILLUSTRATOR CC SCRIPTING GUIDE

Adobe Illustrator CC Scripting Guide

Contents

1 Introduction

This guide describes the scripting interface to Adobe® Illustrator® CC.

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.

Scripting language support in Adobe Illustrator CC

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

Script file extensions

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

Script type	File type (extension)	Platforms
AppleScript	compiled script (`.scpt`) OSAS file (no extension)	Mac OS
JavaScript or ExtendScript	text (`.js` or `.jsx`)	Windows Mac OS
VBScript	text (`.vbs`)	Windows

JavaScript development options

You can use the ExtendScript Toolkit to create JavaScript scripts explicitly for Illustrator, or you can use Adobe Extension Builder and the Creative Cloud SDK to develop *extensions* in ActionScript. Extensions are Flash-based (SWF) and can potentially work in a variety of Creative Cloud applications.

Developing a CC extension using ActionScript

Creative Cloud applications have an extensibility infrastructure that allows developers to extend the capabilities of the applications; the infrastructure is based on Flash/Flex technology, and each extension is delivered as compiled Flash (SWF) file. Creative Cloud includes the Extension Manager to enable installation of extensions.

An example of an extension that ships with the point products is Adobe Kuler. Kuler has a consistent user interface across the different suite applications, but has different logic in each, adapted to the host application.

The user interface for an extension is written in ActionScript, using the Flex framework. An extension is typically accessed through its own menu item in the application's Extensions menu. Adobe Extension Builder allows you to design the user interface interactively using the Design view of Flash Builder. The Creative Cloud SDK also allows you to develop all of the application logic for your extension in ActionScript; you can develop and debug your extension in the familiar Flash Builder environment.

To develop your application logic, we recommend using the ActionScript Wrapper Library (`CSAWLib`), which exposes the scripting DOM of each host application as an ActionScript library. This is tightly integrated with the Adobe Extension Builder environment, which includes wizards to help you build your extension's basic structure, and run and debug your code against suite applications such as Adobe InDesign, Photoshop and Illustrator.

The methods, properties, and behavior of the scripting DOM is as described in the *JavaScript Scripting Reference* for the host application. For details of how to use Adobe Extension Builder and the wrapper

libraries, see the Creative Cloud SDK documentation, which is accessible from within Adobe Extension Builder.

Scripting plug-ins

The CC JavaScript scripting interface allows for limited scripting for plug-ins. A plug-in can define a command, with an event and notifier, and a handler that performs some action. A JavaScript script can then use the `app.sendScriptMessage()` method to send parameters to that plug-in-defined command, and receive a plug-in-defined response.

For example, the Adobe Custom Workspace plug-in defines a command "Switch Workspace". A script can invoke this command with the following code:

```
result = app.sendScriptMessage("Adobe Custom Workspace" ,
          "Switch Workspace", '<workspace="Essentials" >');
```

In this case, the value that the plug-in returns is the string "<error=*errNo*>".

ExtendScript features

If you write Illustrator-specific scripts that use the Illustrator JavaScript DOM directly, you will create ExtendScript files, which are distinguished by the `.jsx` extension. Giving your JavaScript files a `.jsx` extension (rather than the standard `.js` extension for a JavaScript file) allows you to take advantage of ExtendScript features and tools.

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, and is the default editor for JSX files. 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 9.

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

All of these features are available whether you use the DOM directly with a JSX file, or indirectly through the ActionScript wrapper library and Adobe Extension Builder. For details of these and other features, see *JavaScript Tools Guide*.

Viewing sample scripts

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

- In the `/Scripting/Sample Scripts` folder in your Illustrator CC installation directory

- In the Adobe Illustrator CC scripting reference for your scripting language, which you can download from http://www.adobe.com/devnet/illustrator/scripting/

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 CC\ExtendScript Toolkit CC
   - Mac OS:
     system drive:Applications:Utilities:Adobe Utilities CC:ExtendScript Toolkit CC

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

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

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

You also can view script samples and information about individual classes, objects, properties, methods, and parameters in *Adobe Illustrator CC Scripting Reference: JavaScript*, which you can download from http://www.adobe.com/devnet/illustrator/scripting/.

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.

2. Choose File > Open Dictionary. Script Editor displays an Open Dictionary dialog.

3. In the Open Dictionary dialog, find and select Adobe Illustrator CC, 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 CC installation directory.

You also can view script samples and information about individual classes, objects, properties, methods and parameters in *Adobe Illustrator CC Scripting Reference: AppleScript*, which you can download from http://www.adobe.com/devnet/illustrator/scripting/.

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 CC 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 CC installation directory.

You also can view script samples and information about individual classes, objects, properties, methods, and parameters in *Adobe Illustrator CC Scripting Reference: VBScript*, which you can download from http://www.adobe.com/devnet/illustrator/scripting/.

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 11.

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 11.

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 CC/Presets` folder in your Illustrator CC 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 "Scripting language support in Adobe Illustrator CC" 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 IllustratorCC
- A general startup scripts folder, which contains scripts that run automatically when you start any Creative Suite 5 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 IllustratorCC is installed to its default location, you would create the `Startup Scripts` folder at the following location:

- Windows: `C:\Program Files\Adobe\Adobe Illustrator CC\Startup Scripts\`
- Mac OS: `/Applications/Adobe Illustrator CC/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).

General startup scripts folder

The general startup scripts folder contains scripts that run automatically when you start any Creative Suite 5 application. You create the folder in the following location:

- Windows: `Program Files/Common Files/Adobe/Startup Scripts CC/Illustrator`
- Mac OS: `:Library:Application Support:Adobe:Startup Scripts CC: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:

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

For details, see *JavaScript Tools Guide*.

Changes in CC

This section lists changes made to the scripting object model to support features in lllustrator CC. Detailed descriptions can be found in the Scripting Reference documents for each scripting language.

Enumerations and constants

- A new enumeration value and default (for AutoCad compatibility):
  - AppleScript — `auto cad compatibility`: new values `auto cad release 21` and `auto cad release 24` (default)
  - JavaScript — `AutoCADCompatibility.AutoCADRelease21` and `AutoCADRelease24` (default)
  - VBScript — `AutoCADCompatibility.aiAutoCADRelease21` and `aiAutoCADRelease24` (default)
- A new value and default for version compatibility when saving in EPS or AI format:
  - AppleScript — `compatibility`: new value `illustrator 17`
  - JavaScript — `Compatibility.Illustrator17`
  - VBScript — `Compatibility.aiIllustrator17`
- A new constant for layout styles of multiple documents:
  - AppleScript — `document layout style`: values `cascade`, `horizontal tile`, `vertical tile`, `float all`, `consolidate all`
  - JavaScript — `DocumentLayoutStyle.CASCADE, HORIZONTALTILE, VERTICALTILE, FLOATALL, CONSOLIDATEALL`
  - VBScript — `DocumentLayoutStyle.aiCASCADE, aiHORIZONTALTILE, aiVERTICALTILE, aiFLOATALL, aiCONSOLIDATEALL`

Methods and properties

- New application methods/commands manipulate workspaces:
  - AppleScript — `save workspace, switch workspace, delete workspace, reset workspace`
  - JavaScript — `saveWorkspace(), switchWorkspace(), deleteWorkspace(), resetWorkspace()`
  - VBScript — `SaveWorkspace(), SwitchWorkspace(), DeleteWorkspace(), ResetWorkspace()`
- A new document method/command specifies a layout style for multiple documents:
  - AppleScript — `arrange [document layout style]`

- JavaScript — `arrange(layoutStyle)`
- VBScript — `Arrange(layoutStyle as DocumentLayoutStyle)`

- New text-frame methods/commands convert between area-type and point-type objects:
  - AppleScript —`convert area object to point object, convert point object to area object`
  - JavaScript — `convertAreaObjectToPointObject(), convertPointObjectToAreaObject()`
  - VBScript — `ConvertAreaObjectToPointObject(), ConvertPointObjectToAreaObject()`
- New SVG export options:
  - AppleScript —`save multiple art boards`, `artboard range`, `include unused styles`
  - JavaScript — `saveMultipleArtBoards`, `artboardRange`, `includeUnusedStyles`
  - VBScript — `SaveMultipleArtBoards`, `ArtboardRange`, `IncludeUnusedStyles`

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

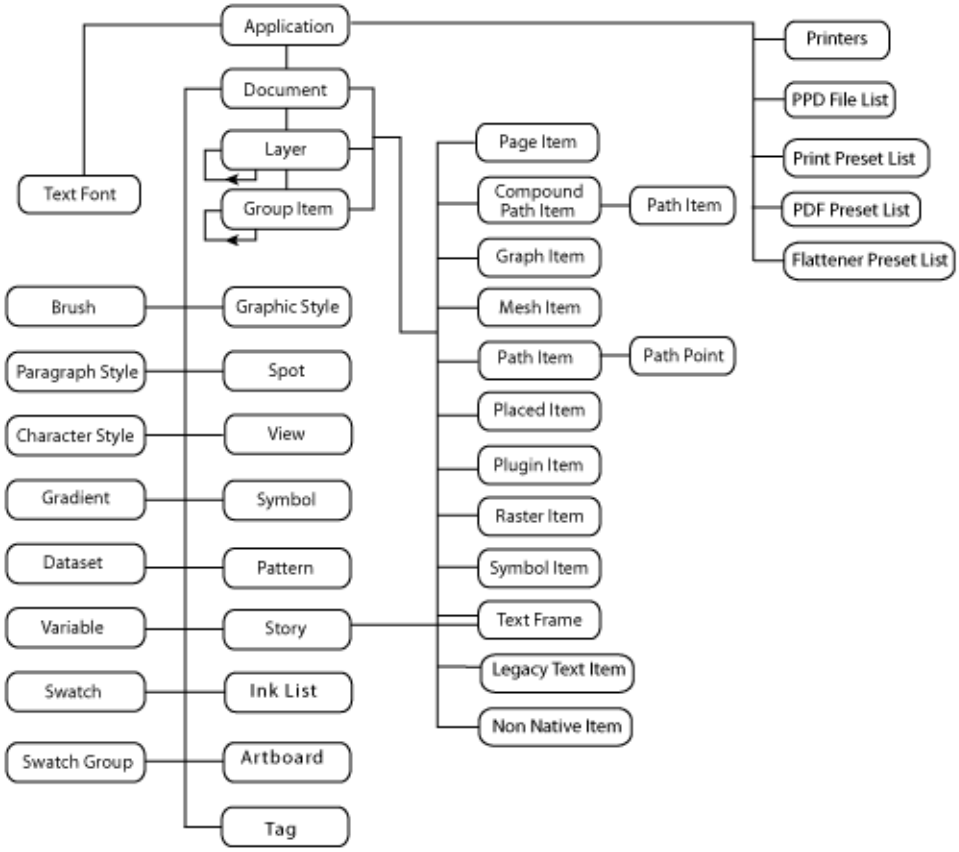
  For more information on quitting and relaunching Illustrator, see "Launching and activating Illustrator" on page 22 and "Quitting Illustrator" on page 23.
- The "An Illustrator error occurred: 1346458189 ("PARM")" alert may be popped when badly written scripts are repeatedly run in Illustrator from the 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.
- If you create more than one art object in AppleScript and assign each one to a variable, all the variables are set to the last item. This means that the previously created items are not accessible.

2 The Illustrator Scripting Object Model

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

- `mesh item`
- `non native item`
- `path item`
- `placed item`
- `plugin item`
- `raster item`
- `symbol item` (see "Dynamic objects" on page 21)
- `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 CC 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.

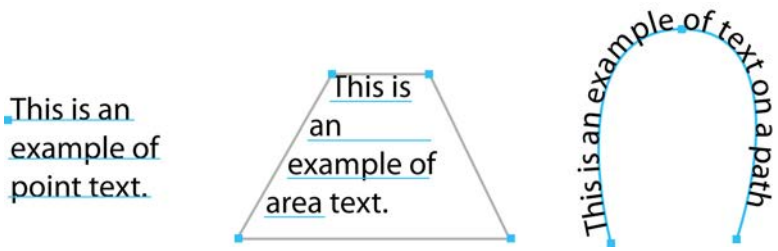
To observe this, open a new document in Illustrator and use the horizontal text tool to type some text, then use the vertical text tool to type more text. Finally, create a rectangle and type some text inside it. Now run the following JavaScript script:

```
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.

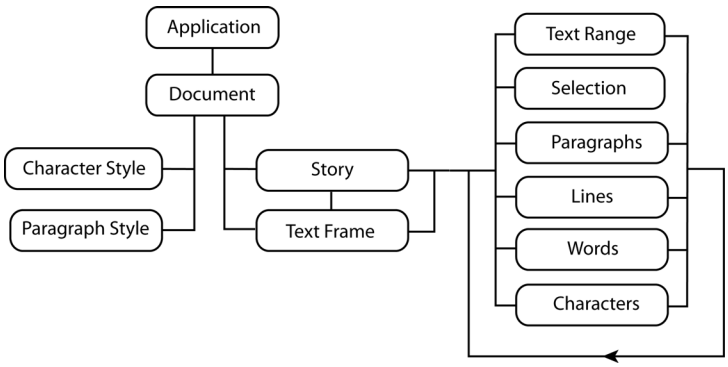
NOTE: A text path is not the same as a path art item. Text paths are associated with path art items that can be accessed and manipulated to modify the appearance of the associated text frame.

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 CC. 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:

  ```
  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"`
  For Illustrator CS5, use `"Illustrator.Application.CS5"`

For Illustrator CS6, use `"Illustrator.Application.CS6"`
For Illustrator CC, use `"Illustrator.Application.CC"`

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

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 CC Scripting Reference: AppleScript* or the Properties table in *Adobe Illustrator CC Scripting Reference: JavaScript* or *Adobe Illustrator CC Scripting Reference: VBScript*. A diagram of the Illustrator CC object model is in "The Illustrator Scripting Object Model" on page 14.

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 26.)

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:

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

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

Page-item positioning and dimensions

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

For information on the types of objects that comprise the `page items` collection, see "The artwork tree" on page 16.

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.

For the artboard, the default coordinate origin, (0,0), is the top-left corner, reflected in the `ruler origin` property of the `artboard` object. X coordinate values increase from left to right, and Y values increase from top to bottom. This changed in the CS5 release; to maintain script compatability, a document created by a script still uses the older system, with the origin at the bottom left of the artboard, and the Y value increasing from bottom to top. The `page origin` property of a `document` object defines the bottom-left corner of the printable region of the document as a fixed point.

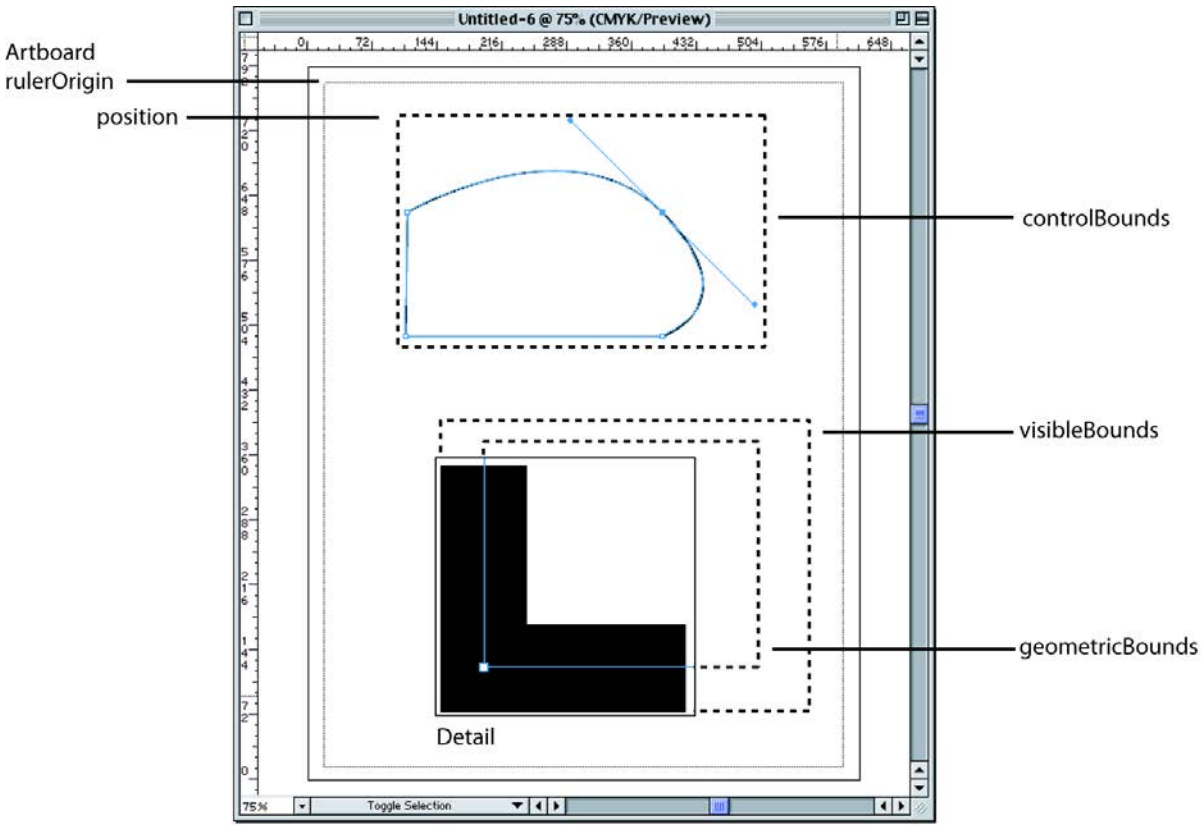
Each `page item` object has `width` and `height` properties. The maximum value allowed for the width or height of a page item is 16348 points.

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.

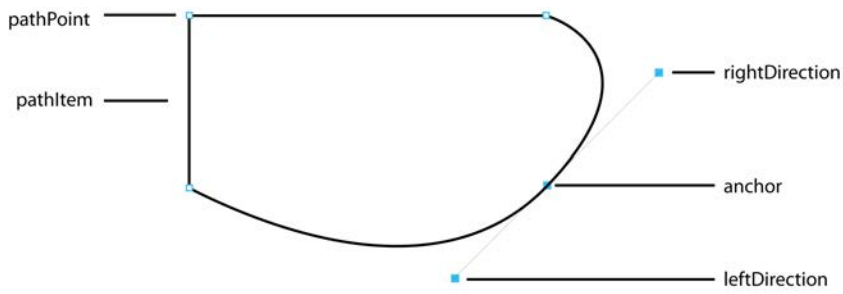


Paths and shapes

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:

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

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

JavaScript

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

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

VBScript

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

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

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.

4 Scripting with AppleScript

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

For more information

Several extended sample scripts are in the `:Scripting:Sample Scripts` folder in your Illustrator CC 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 CC Scripting Reference: AppleScript*, in the `:Scripting:Documentation` folder in your Illustrator CC installation directory. You also can view the Illustrator CC dictionary from the Script Editor application;see "Viewing the AppleScript 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!" 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:

```
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:

```
-- 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}
set myAreaText to make new text frame in current document with properties ?
{kind:point text,contents:"Text Frame 1"}
```

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

```
      text frame "tf2" of current document
   redraw
end tell
```

Threaded frames make one story object

Threaded frames make a single `story` object. To observe this, run the following AppleScript after running the script in "Threaded frames" on page 33.

```
display dialog ("There are " & (count(text frames of current document)) & " text frames.")
display dialog("There are " & (count(stories of current document)) & " stories.")
```

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 `path point` objects.

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

Specifying a series of x-y coordinates

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

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

Using path point objects

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

- A fixed `anchor` point, which is the point on the path.
- A pair of direction points—`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 `path point` object have the same coordinates, and the properties of the next `path point` in the line are equal to each other, you create a straight-line segment.
- If two or more properties in a `path point` object have different values, the segment connected to the point is curved.

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

```
tell application "Adobe Illustrator"
set docRef to make new document
-- set stroked to true so we can see the path
set lineRef to make new path item in docRef with properties {stroked:true}
    --giving the direction points the same value as the
    --anchor point creates a straight line segment
set newPoint to make new path point of lineRef with properties ?
    {anchor:{220, 475},left direction:{220, 475},right direction:{220, 475},
    point type:corner}

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

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

end tell
```

Combining path point types

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

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

Shapes

To create a shape, you use the object that corresponds to the shape's name (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 26.
- 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 26.

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

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 is 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).

Working with the perspective grid

The Perspective Grid is a new feature in lllustrator CC that enables you to create and manipulate art in a spatial environment using established laws of perspective. Enable Perspective Grid using the View > Perspective Grid menu or the perspective tools in the toolbar.

The SDK provides an API for working with the perspective grid programmatically, and your scripts have some access to this API. A script can:

- Set a the default grid parameters using preset values.

- Show or hide the grid.
- Set the active plane.
- Draw an object in perspective on the active plane.
- Bring an object into perspective.

Use perspective presets

Illustrator provides default grid-parameter presets for one-point, two-point, and three-point perspectives. The presets are named `"[1P-NormalView]"`, `"[2P-NormalView]"`, and `"[3P-NormalView]"`.

The script shows how to select the two-point perspective preset programmatically:

```
tell application "Adobe Illustrator"
    --Create a new document
    set docRef to make new document
    tell docRef
        --Select the default two-point perspective preset
        select perspective preset perspective preset "[2P-Normal View]"
    end tell
end tell
```

You can create new perspective presets, export presets to files, and import presets from files. These scripts shows how to export and import presets:

```
tell application "Adobe Illustrator"
    set docRef to make new document
    set filePath to "Macintosh HD:scripting:PGPresetsExported"
    export perspective grid preset of docRef to file filePath
end tell

tell application "Adobe Illustrator"
    set docRef to make new document
    set filePath to "Macintosh HD:scripting:PGPresets"
    import perspective grid preset of docRef from file filePath
end tell
```

Show or hide the grid

This script shows or hides the Perspective Grid programmatically:

```
tell application "Adobe Illustrator"
    --Create a new document
    set docRef to make new document
    tell docRef
        --Display the perspective grid defined in the document
        show perspective grid
        --Hide the perspective grid defined in the document
        hide perspective grid
    end tell
end tell
```

Set the active plane

The perspective grid plane types are:

Left plane	`perspective grid plane leftplane`
Right plane	`perspective grid plane rightplane`
Floor plane	`perspective grid plane floorplane`
Invalid plane	`perspective grid plane noplane`

For a one-point perspective grid, only the left and floor plane are valid.

This script sets the active perspective plane to the left plane:

```
tell application "Adobe Illustrator"
    --Create a new document
    set docRef to make new document
    tell docRef
        --Set the active plane to the left plane
        set perspective active plane perspective grid plane leftplane
    end tell
end tell
```

Draw on a perspective grid

When the Perspective Grid is on, drawing methods allow you to draw or operate on objects in perspective. This script creates a new document, shows a two-point perspective grid, and draws art objects on the left plane:

```
tell application "Adobe Illustrator"
    --Create a new document
    set docRef to make new document
    tell docRef
        --Select the default two-point perspective preset
        select perspective preset perspective preset "[2P-Normal View]"

        --Display the perspective grid defined in the document
        show perspective grid

        --Check if active plane is set to left, otherwise set it to left
        if (get perspective active plane) is not leftplane then
            set perspective active plane perspective grid plane leftplane
        end if

        --Draw rectangle in perspective, then resize to 200% and move
        set rectRef to make new rectangle with properties {bounds:{0, 0, 30, 30},
reversed:false}
        scale rectRef horizontal scale 200 vertical scale 200 about top left with
transforming objects
        translate rectRef delta x -420 delta y 480

        --Draw ellipse in perspective
        set ellipseRef to make new ellipse with properties {bounds:{60, -60, 90, -30},
reversed:false, inscribed:true}
```

```
        --Draw rounded rectangle in perspective
        set rrectRef to make new rounded rectangle with properties {bounds:{90, -90, 30,
30}, horizontal radius:10, vertical radius:10, reversed:false}

        --Draw polygon in perspective
        set polyRef to make new polygon with properties {center point:{105, 105},
radius:15, sides:7, reversed:false}

        --Draw star in perspective
        set starRef to make new star with properties {center point:{135, 135},
radius:15, inner radius:10, point count:6, reversed:false}

        --Draw path in perspective
        set newPath to make new path item with properties {entire path:{{anchor:{0, 0}},
{anchor:{60, 0}}, {anchor:{30, 45}}, {anchor:{90, 110}}}}
    end tell
end tell
```

Bring objects into perspective

If an art object is not in perspective, use the `bringInPerspective()` method to bring it into perspective and place it on a plane.

This script creates a new document, draws an art object, and brings it into perspective on a three-point perspective grid:

```
tell application "Adobe Illustrator"
    --Create a new document
    set docRef to make new document
    tell docRef
        --Draw star
        set starRef to make new star with properties {center point:{135, 135},
radius:15, inner radius:10, point count:6, reversed:false}

        --Select the default three-point perspective preset
        select perspective preset perspective preset "[3P-Normal View]"

        --Display the perspective grid defined in the document
        show perspective grid

        --Check if active plane is set to left, otherwise set it to left
        if (get perspective active plane) is not leftplane then
            set perspective active plane perspective grid plane leftplane
        end if

        --Bring star to floor plane
        bring in perspective starRef position x 100 position y 100 perspective grid plane
floorplane
    end tell
end tell
```

5 Scripting with JavaScript

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

For more information

Several extended sample scripts are in the `/Scripting/Sample Scripts` folder in your Illustrator CC 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 CC Scripting Reference: JavaScript*, in the `/Scripting/Documentation` folder in your Illustrator CC 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 9 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:

```
//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 9.

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

- If you are using the ESTK, select Adobe Illustrator CC 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:

```
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 CC 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.

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 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:

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

- Using the containment hierarchy:

```
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:

```
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:

```
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:

```
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 `text frame` object.

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

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

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

Threaded frames make a single story object

Threaded frames make a single `story` object. To observe this, run the following JavaScript after running the script in "Threaded frames" on page 44.

```
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:

```
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.stroked = true;

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

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

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

Combining path point types

The following script sample creates a path with three points:

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

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

Shapes

To create a shape, 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 26.
- 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 26.

Creating a rectangle

Consider the following sample

```
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:

```
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.

Working with the perspective grid

The Perspective Grid is a new feature in lllustrator CC that enables you to create and manipulate art in a spatial environment using established laws of perspective. Enable the Perspective Grid using the View > Perspective Grid menu or the perspective tools in the toolbar.

The SDK provides an API for working with the perspective grid programmatically, and your scripts have some access to this API. A script can:

- Set a the default grid parameters using preset values.
- Show or hide the grid.
- Set the active plane.
- Draw an object in perspective on the active plane.
- Bring an object into perspective.

Use perspective presets

Illustrator provides default grid-parameter presets for one-point, two-point, and three-point perspectives. The presets are named `"[1P-NormalView]"`, `"[2P-NormalView]"`, and `"[3P-NormalView]"`.

This script shows how to select a preset programmatically:

```
//Set the default one-point perspective preset
app.activeDocument.selectPerspectivePreset("[1P-Normal View]");

//Set the default two-point perspective preset
app.activeDocument.selectPerspectivePreset("[2P-Normal View]");

//Set the default three-point perspective preset
app.activeDocument.selectPerspectivePreset("[3P-Normal View]");
```

You can create new perspective presets, export presets to files, and import presets from files. These scripts shows how to export and import presets:

```
//Create a new document
var mydoc = app.documents.add();
//Export perspective presets to a file
var exportPresetFile = new File("C:/scripting/PGPresetsExported")
mydoc.exportPerspectiveGridPreset(exportPresetFile);

//Create a new document
var mydoc = app.documents.add();
//Import perspective presets from a file
var importPresetFile = new File("C:/scripting/PGPresets")
mydoc.importPerspectiveGridPreset(importPresetFile);
```

Show or hide the grid

This script shows or hides the Perspective Grid programmatically:

```
//Show the Perspective Grid defined in the document
app.activeDocument.showPerspectiveGrid();

//Hide the Perspective Grid defined in the document
mydoc.hidePerspectiveGrid();
```

Set the active plane

The perspective grid plane types are:

Left plane	`PerspectiveGridPlaneType.LEFTPLANE`
Right plane	`PerspectiveGridPlaneType.RIGHTPLANE`
Floor plane	`PerspectiveGridPlaneType.FLOORPLANE`
Invalid plane	`PerspectiveGridPlaneType.NOPLANE`

For a one-point perspective grid, only the left and floor plane are valid.

This script sets the active perspective plane:

```
//Set left plane as the active plane
app.activeDocument.setPerspectiveActivePlane(PerspectiveGridPlaneType.LEFTPLANE);

//Set right plane as the active plane
app.activeDocument.setPerspectiveActivePlane(PerspectiveGridPlaneType.RIGHTPLANE);

//Set floor plane as the active plane
app.activeDocument.setPerspectiveActivePlane(PerspectiveGridPlaneType.FLOORPLANE);
```

Draw on a perspective grid

When the Perspective Grid is on, drawing methods allow you to draw or operate on objects in perspective. This script creates a new document, shows a two-point perspective grid, and draws art objects on the left plane:

```
//Create a new document
var mydoc = app.documents.add();

//Select the default two-point perspective preset
mydoc.selectPerspectivePreset("[2P-Normal View]");

//Display the perspective grid defined in the document
mydoc.showPerspectiveGrid();

//Check if active plane is set to left; if not, set it to left
if (mydoc.getPerspectiveActivePlane() != PerspectiveGridPlaneType.LEFTPLANE)
{
    mydoc.setPerspectiveActivePlane(PerspectiveGridPlaneType.LEFTPLANE);
}

//Draw rectangle in perspective, then resize to 200% and move
var myrect = mydoc.pathItems.rectangle(30, -30, 30, 30, false);
myrect.resize(200, 200, true, false, false, false, 100, Transformation.TOPLEFT);
myrect.translate (-420, 480);

//Draw ellipse in perspective
var myellipse = mydoc.pathItems.ellipse(60, -60, 30, 30, false, true);

//Draw rounded rectangle in perspective
var myrrect = mydoc.pathItems.roundedRectangle(90, -90, 30, 30, 10, 10, false);

//Draw polygon in perspective
var mypoly = mydoc.pathItems.polygon(-105, 105, 15, 7, false);

//Draw star in perspective
var mystar = mydoc.pathItems.star(-135, 135, 15, 10, 6, false);

//Draw path in perspective
var newPath = mydoc.pathItems.add();
var lineList = new Array(4);
lineList[0] = new Array(0,0);
lineList[1] = new Array(60,0);
lineList[2] = new Array(30,45);
lineList[3] = new Array(90,110);
newPath.setEntirePath(lineList);
```

Bring objects into perspective

If an art object is not in perspective, use the `bringInPerspective()` method to bring it into perspective and place it on a plane.

This script creates a new document, draws art objects, and brings them into perspective on a three-point perspective grid:

```
ï»¿//Create a new document
var mydoc = app.documents.add();

//Draw ellipse
var myellipse = mydoc.pathItems.ellipse(60, -60, 30, 30, false, true);

//Draw polygon
var mypoly = mydoc.pathItems.polygon(-105, 105, 15, 7, false);

//Draw star
var mystar = mydoc.pathItems.star(-135, 135, 15, 10, 6, false);

//Select the default three-point perspective preset
mydoc.selectPerspectivePreset("[3P-Normal View]");

//Display the perspective grid defined in the document
mydoc.showPerspectiveGrid();

//Check if active plane is set to left; if not, set it to left
if (mydoc.getPerspectiveActivePlane() != PerspectiveGridPlaneType.LEFTPLANE)
{
    mydoc.setPerspectiveActivePlane(PerspectiveGridPlaneType.LEFTPLANE);
}

//Bring the ellipse to the active plane (left plane)
myellipse.bringInPerspective(-100,-100, PerspectiveGridPlaneType.LEFTPLANE);

//Bring the polygon to the right plane
mypoly.bringInPerspective(100,-100,PerspectiveGridPlaneType.RIGHTPLANE);

//Bring the star to the floor plane
mystar.bringInPerspective(100,100,PerspectiveGridPlaneType.FLOORPLANE);
```

6 Scripting with VBScript

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

For more information

Several extended sample scripts are in the `/Scripting/Sample Scripts` folder in your Illustrator CC 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 CC Scripting Reference: VBScript*, in the `/Scripting/Documentation` folder in your Illustrator CC installation directory. You also can view the Illustrator CC type library from most VBScript editors or any Microsoft Office application; see "Viewing the VBScript object model" on page 10.

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:

```
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.

```
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:

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

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

```
Set documentRef = appRef.ActiveDocument

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

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

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

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:

```
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`:

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

Working with selections

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

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

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

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

```
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:

```
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.

```
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 54.

```
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) & " storiess. "")"
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:

```
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`:

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

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

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

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

Combining path-point types

The following script sample creates a path with three points:

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

' Append another point to the line
Set newPoint = 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 26.
- 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 26.

Creating a rectangle

Consider the following sample:

```
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 10.
- The *Adobe Illustrator CC 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:

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

The following sample specifies vertical text orientation:

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

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
```

Working with the perspective grid

The Perspective Grid is a new feature in Illustrator CC that enables you to create and manipulate art in a spatial environment using established laws of perspective. Enable the Perspective Grid using the View > Perspective Grid menu or the perspective tools in the toolbar.

The SDK provides an API for working with the perspective grid programmatically, and your scripts have some access to this API. A script can:

- Set a the default grid parameters using preset values.
- Show or hide the grid.
- Set the active plane.
- Draw an object in perspective on the active plane.
- Bring an object into perspective.

Use perspective presets

Illustrator provides default grid-parameter presets for one-point, two-point, and three-point perspectives. The presets are named `"[1P-NormalView]"`, `"[2P-NormalView]"`, and `"[3P-NormalView]"`.

The script shows how to select the two-point perspective preset programmatically:

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

Rem Create a new document
Set docRef = appRef.Documents.Add()

Rem Select the default two-point perspective preset
docRef.SelectPerspectivePreset("[2P-Normal View]")
```

You can create new perspective presets, export presets to files, and import presets from files. These scripts shows how to export and import presets:

```
Set appRef = CreateObject ("Illustrator.Application")
Rem Create a new document
Set docRef = appRef.Documents.Add()
Rem Export perspective presets to a file
docRef.ExportPerspectiveGridPreset("C:/scripting/PGPresetsExported")


Set appRef = CreateObject ("Illustrator.Application")
Rem Create a new document
Set docRef = appRef.Documents.Add()
Rem Import perspective presets from a file
docRef.ImportPerspectiveGridPreset("C:/scripting/PGPresets")
```

Show or hide the grid

This script shows or hides the Perspective Grid programmatically:

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

Rem Create a new document
```

```
Set docRef = appRef.Documents.Add()

Rem Show the Perspective Grid defined in the document
docRef.ShowPerspectiveGrid();

Rem Hide the Perspective Grid defined in the document
docRef.HidePerspectiveGrid();
```

Set the active plane

The perspective grid plane types are:

Left plane	`aiLEFTPLANE (1)`
Right plane	`aiRIGHTPLANE (2)`
Floor plane	`aiFLOORPLANE (3)`
Invalid plane	`aiNOPLANE (4)`

For a one-point perspective grid, only the left and floor plane are valid.

This script sets the active perspective plane to the left plane:

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

Rem Create a new document
Set docRef = appRef.Documents.Add()

Rem Set left plane as the active plane
docRef.SetPerspectiveActivePlane(1) 'aiLEFTPLANE
```

Draw on a perspective grid

When the Perspective Grid is on, drawing methods allow you to draw or operate on objects in perspective. This script creates a new document, shows a two-point perspective grid, and draws art objects on the left plane:

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

Rem Create a new document
Set docRef = appRef.Documents.Add()

Rem Select the default two point perspective preset
docRef.SelectPerspectivePreset("[2P-Normal View]")

Rem Display the perspective grid defined in the document
docRef.ShowPerspectiveGrid()

Rem Check if active plane is set to left, otherwise set it to left
If docRef.GetPerspectiveActivePlane() <> 1 Then
   docRef.SetPerspectiveActivePlane(1) 'aiLEFTPLANE
End If

Rem Draw rectangle in perspective, then resize to 200% and move
Set pathItemRect = docRef.PathItems.Rectangle(30, -30, 30, 30, False)
```

```
call pathItemRect.Resize(200, 200, True, False, False, False, 100, 2)
call pathItemRect.Translate(-420, 480)

Rem Draw ellipse in perspective
Set pathItemEllipse = docRef.PathItems.Ellipse(60, -60, 30, 30, False, True)

Rem Draw rounded rectangle in perspective
Set pathItemRRect = docRef.PathItems.RoundedRectangle(90, -90, 30, 30, 10, 10, False)

Rem Draw polygon in perspective
Set pathItemPoly = docRef.PathItems.Polygon(-105, 105, 15, 7, False)

Rem Draw star in perspective
Set pathItemStar = docRef.PathItems.Star(-135, 135, 15, 10, 6, False)

Rem Draw path in perspective
Set newPath = docRef.PathItems.Add()
newPath.SetEntirePath(Array(Array(0,0),Array(60,0),Array(30,45),Array(90,110)))
```

Bring objects into perspective

If an art object is not in perspective, use the `bringInPerspective()` method to bring it into perspective and place it on a plane.

This script creates a new document, draws art objects, and brings them into perspective on a three-point perspective grid:

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

Rem Create a new document
Set docRef = appRef.Documents.Add()

Rem Draw ellipse
Set pathItemEllipse = docRef.PathItems.Ellipse(60, -60, 30, 30, False, True)

Rem Draw polygon
Set pathItemPoly = docRef.PathItems.Polygon(-105, 105, 15, 7, False)

Rem Draw star
Set pathItemStar = docRef.PathItems.Star(-135, 135, 15, 10, 6, False)

Rem Select the default three-point perspective preset
docRef.SelectPerspectivePreset("[3P-Normal View]")

Rem Display the perspective grid defined in the document
docRef.ShowPerspectiveGrid()

Rem Check if active plane is set to left, otherwise set it to left
If docRef.GetPerspectiveActivePlane() <> 1 Then
   docRef.SetPerspectiveActivePlane(1) 'aiLEFTPLANE
End If

Rem Bring the ellipse to the active plane (left plane)
Call pathItemEllipse.BringInPerspective(100,100, 1) 'aiLEFTPLANE

Rem Bring the polygon to the right plane
Call pathItemPoly.BringInPerspective(100,-100,2) 'aiRIGHTPLANE
```

```
Rem Bring the star to the floor plane
Call pathItemStar.BringInPerspective(100,100,3) 'aiFLOORPLANE
```

Index

W

X

Y

Z