ADOBE® ILLUSTRATOR®

GETTING STARTED WITH ADOBE ILLUSTRATOR 2019 DEVELOPMENT

Getting Started with Adobe Illustrator 2019 Development

Technical Note #10501

Contents

Getting Started with Adobe Illustrator 2019 Development

This document describes how to start developing plug-ins for Adobe® Illustrator® 2019, using the Software Development Kit (SDK). The target audience for this document is developers who need to develop plug-ins for Illustrator.

Preface

The Illustrator 2019 SDK is available for download from:

https://console.adobe.io/downloads/ai

Using this document

To start developing plug-ins for Illustrator, follow these steps:

1. Set up your machine for development. Follow the instructions in "Setting up your development platform" on page 6.
2. Explore the documentation; see "Exploring the documentation" on page 7.
3. Explore, compile, and run the samples; see "Exploring the samples" on page 8.
4. Continue exploring the API using *code snippets*; see "Exploring the API using code snippets" on page 17.
5. Write a new plug-in using the supplied SDK plug-in project templates; see "Using the plug-in project templates" on page 25.
6. Write a "HelloWorld" plug-in from scratch to learn about configuring your development environment; see "Creating a plug-in" on page 9.

Terminology and conventions

- `<SDK>` refers to your locally installed SDK root folder. The actual root location depends on the installation and operating system.
- `<AI>` refers to the installed location of Illustrator 2019. The actual location depends on the installation and operating system.
- `<VS>` indicates the installed location of Microsoft Visual Studio 2017. The actual location depends on the installation; the default is `C:\Program Files (x86)\Microsoft Visual Studio\2017`

Setting up your development platform

1. Check that your platform meets the basic requirements:

Platform	Component	
Windows	Windows 7 with Service Pack 1 or higher	
	Visual Studio 2017 Microsoft Visual C++ 2017	Developing Illustrator 2019 plug-ins in Windows requires Visual C++, a component of Visual Studio 2017. If you have the Professional or Standard Editions of Visual Studio 2013, you must install the Visual C++ components.
Mac OS	Mac OS 10.12 or higher	
	Xcode 9.2 LLVM Clang (`com.apple.compilers.llvm.clang.1_0`) Base SDK: OS X 10.13 OS X Deployment Target: OS X 10.12	Xcode can be downloaded from http://developer.apple.com/tools/download/

2. Download the Illustrator 2019 SDK for your platform from:

   https://console.adobe.io/downloads/ai

   ▷ In Windows, extract the contents of the downloaded zip archive to a location of your choice.

   ▷ In Mac OS, mount the downloaded disk image file (DMG) as a new volume, and copy the Illustrator 2019 SDK folder to a location of your choice.

3. For detailed instructions on configuring the development environment for plug-in creation, see:

   ▷ "Creating a plug-in in Windows" on page 28

   ▷ "Creating a plug-in in Mac OS" on page 32

Exploring the SDK

The SDK contains documentation and code samples to help you in developing your own Illustrator 2019 plug-ins, and in porting plug-ins from previous releases.

Exploring the documentation

The following documents are included with the SDK:

Title	Description
Adobe Illustrator 2019 Programmer's Guide	Describes the basic concepts of developing plug-ins for Illustrator 2019. It is aimed at all developers and is the recommended resource for plug-in developers after reading this document. This document is in the file `<SDK>/docs/guides/programmers-guide.pdf`
Adobe Illustrator 2019 Porting Guide	Describes how to update 2014 SDK plug-in code for Illustrator 2019; it is aimed at developers with pre-existing plug-ins. It lists new features and changes in the API since the previous release. This document is in the file `<SDK>/docs/guides/porting-guide.pdf`.
Using the Adobe Text Engine	Describes how to configure your plug-in to use the Adobe text engine API, provided with the Illustrator 2019 SDK. It describes procedures for creating, editing, deleting, styling, and iterating text, with references to sample code and the *API Reference*. This document is in the file `<SDK>/docs/guides/using-adobe-text-engine.pdf`.
API Reference	Provides reference documentation for the suites, classes, structures, functions, variables, and constants available in the API. It is a key resource for all developers. This document is provided in two formats: ▶ `<SDK>/docs/references/index.chm` — This compiled HTML file allows text searches to be performed on the content. ▷ To view the contents in Windows, double-click the `index.chm` file icon in Windows Explorer to open the home page. ▷ To view the contents on a Macintosh, you need a CHM viewer; see Document viewers. ▶ `<SDK>/docs/references/sdkdocs.tar.gz` — This file contains the *API Reference* in HTML format. This format does not support text searches, but it provides the ability to view individual HTML files if desired. To view the contents, first decompress the archive, then open `index.html` in your browser.
API Advisor	Reports the class, structure, and file differences between the API included in the 2018 SDK and 2019 SDK. It is aimed at all developers using the API, but it is most useful for developers updating plug-in code for Illustrator 2019. This document is in the file `<SDK>/docs/references/apiadvisor-ai18-vs-ai19.html`.

Document viewers

- To view PDF-based documentation, you need Adobe Reader or Adobe Acrobat.
- To view HTML-based documentation, you need a web browser.
- To view compiled HTML documentation (CHM) in Mac OS, you need a CHM viewer. Two such viewers are `chmox` (http://chmox.sourceforge.net/) and `xCHM` (http://sourceforge.net/projects/xchm/).

Exploring the samples

The samples provided show how to implement key Illustrator plug-in features like tools, filters, menus, and file formats. The following table shows which features are implemented by each sample:

Feature	Annotator	DrawArt	EmptyPanel	FreeGrid	LiveDropShadow	MarkedObjects	MenuPlay	MultiArrowTool	ScriptMessage	SnippetRunner	StrokeFilter	TextFileFormat	TransformButtons	Tutorial	TwirlFilter	Webter
Action										X				X		
AGM Port		X														
Annotator	X							X								
File Format												X				
Filter											X				X	
Flash/Flex UI		X		X		X			X	X	X		X	X		X
Effect															X	
Menu Item	X		X	X	X	X	X	X		X	X	X	X	X	X	X
Notifier	X					X							X			X
Panel			X			X				X			X			X
Plug-in Group					X											
Scripting									X							
Theme Sync						X		X	X							
Timer										X						
Tool Palette	X					X		X						X		
Transform				X									X			

Use this table to identify the sample most suitable as a starting point for your own plug-in: the sample that implements the most features your plug-in requires will give you the best base from which to work.

- For a more detailed description of each sample, see "Samples guide" on page 10
- For a more detailed description of these plug-in features and what they do, see the "Types of Plug-ins" section in *Adobe Illustrator 2019 Programmer's Guide*.

Creating a plug-in

The SDK samples are based on the `Plugin` and `Suites` classes. The `Plugin` class constructs the basic requirements for an Illustrator plug-in, using the `Suites` class to acquire and release the required suites.

To create your own plug-in using this framework:

- Create a new project, and add `Plugin.cpp` and `Suites.cpp` to your new project from `<SDK>/samplecode/common/source/`.
- Create a class in your project that is a subclass of the `Plugin` class. This class should implement the required functionality of the plug-in.

For a step-by-step example of configuring the development environment and creating a basic plug-in, see:

- "Creating a plug-in in Windows" on page 28
- "Creating a plug-in in Mac OS" on page 32

Building and running the samples

To build all the samples, use the projects provided in the `<SDK>/samplecode/MasterProjects/` folder. After compiling and linking, the plug-in binaries can be found in the `<SDK>/samplecode/Output/` folder.

We recommend you use the Additional Plug-ins Folder preference when developing and debugging plug-ins. Set this preference in the application's Preferences > Plug-ins & Scratch Disk dialog.

For a complete list of the locations where Illustrator looks for plug-ins, see *Adobe Illustrator 2019 Programmer's Guide*.

To debug a sample (or your own plug-in), see "Debugging plug-ins" on page 22.

Samples guide

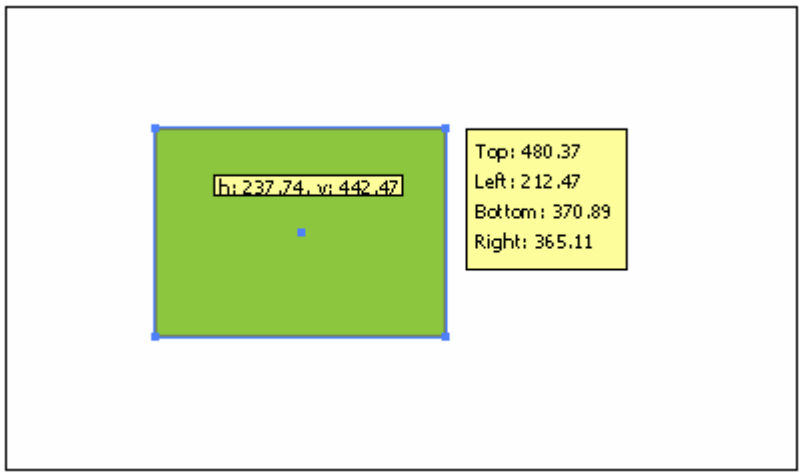
This section describes the individual samples in more detail.

Annotator

Uses the `AIAnnotatorSuite` and `AIAnnotatorDrawerSuite` to annotate artwork in the document and update the annotations as selection or cursor position changes.

Adds a tool to the toolbar, using this icon: 

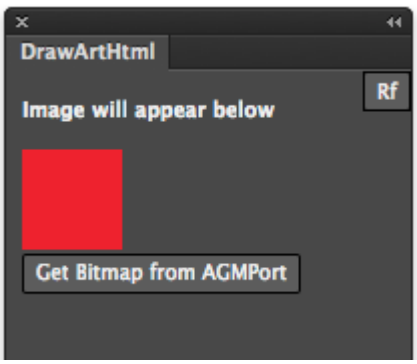
When the annotator tool is selected and the cursor is over an item on the artboard, the item becomes selected, an annotation displaying the bounds of the art appears at the right-hand side of the art item. A smaller annotation displaying the cursor position appears next to the cursor, which updates as the cursor moves within the art.



DrawArt

The DrawArt sample demonstrates how to use the `AIDrawArtSuite` to obtain preview bitmap images to display in Flash panels. The sample draws a swatch to an AGM port, retrieves a bitmap from that port, and passes the bitmap to a Flash panel for display as an image.

This sample is a CS Extension; to run it, choose Window > Extensions > DrawArt.

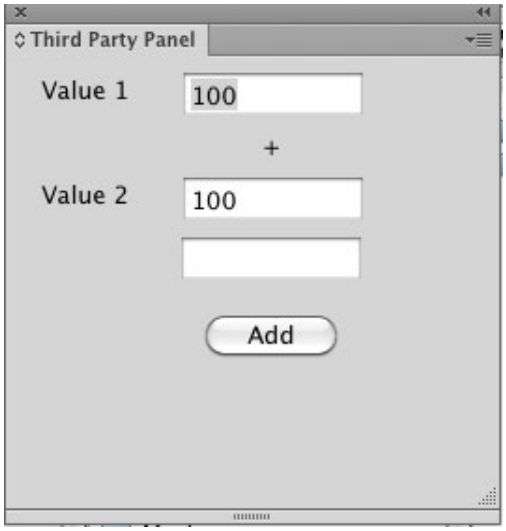


EmptyPanel

The EmptyPanel sample demonstrates how to use native Windows or Mac OS components in a control bar or panel, combining native controls with bars and panels created with `AIControlBarSuite` and `AIPanelSuite`. The bar and panel that the sample creates contain controls that are implemented using native window components.

In the panel, the native controls implement a simple calculator. In the control bar, the values in the native controls are tied to the dimensions of the container, allowing you to resize the control bar dynamically.

The Mac OS examples demonstrate loading a native NIB or XIB file into a panel, so that you can use all the standard Xcode Interface Builder features, and mix objective-C and C++.



Panel and Control Bar using native Mac OS components

Panel and Control Bar using native Windows components

FreeGrid

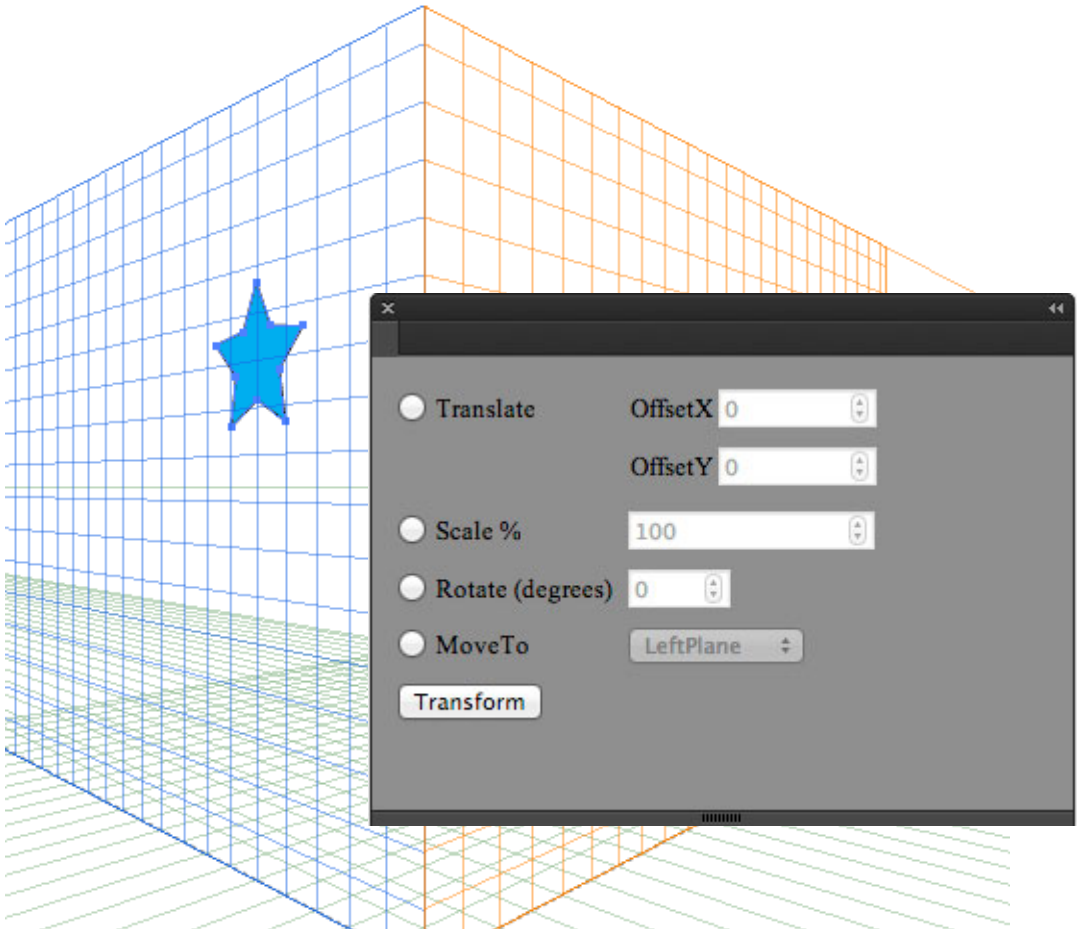
The FreeGrid sample demonstrates the usage of the `AIPerspectiveGridSuite` and `AIPerspectiveTransformSuite` API to manipulate the Perspective Grid, a feature that was added in CS5. `AIPerspectiveGridSuite` allows you to query and set parameters of the currently defined Perspective Grid in a document, and to convert points between the Artwork and Perspective Grid coordinate systems. `AIPerspectiveTransformSuite` allows you to project or reverse-project points and art objects according to the Perspective Grid.

The FreeGridUI extension provides a user interface for the FreeGrid sample, demonstrating how to create a UI for a plug-in using the Flash/Flex-based Creative Suite Extension Builder, which is recommended as a replacement for the deprecated ADM UI solution.

To use this plug-in, first show perspective grid by choosing View > Perspective Grid > Show Grid or by selecting the Perspective Grid Tool in the Tools panel. Draw or select a path art object on the active plane, then choose Window > SDK > FreeGrid.

The resulting dialog allows you to perform one of four operations:

- Translate the art object's position by specifying X and Y offsets.
- Scale the art object's size by specifying a zoom percentage.
- Rotate the art object by specifying an angle.
- Move the art object from one perspective plane to another.

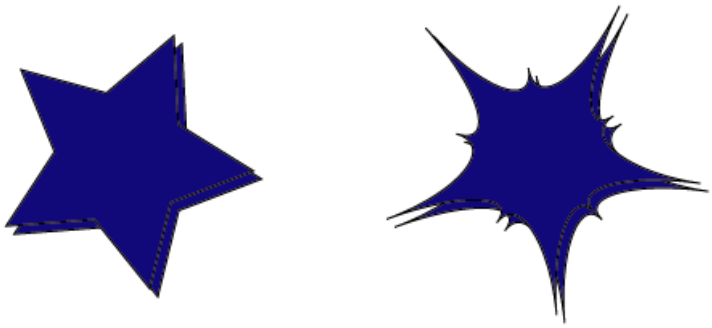


LiveDropShadow

Uses the `Plugin` group suite. It creates "live" drop shadows on objects; these work like the DropShadow filter, except the shadow follows any edits to the object.

Adds the menu item Object > SDK > Live Drop Shadow.

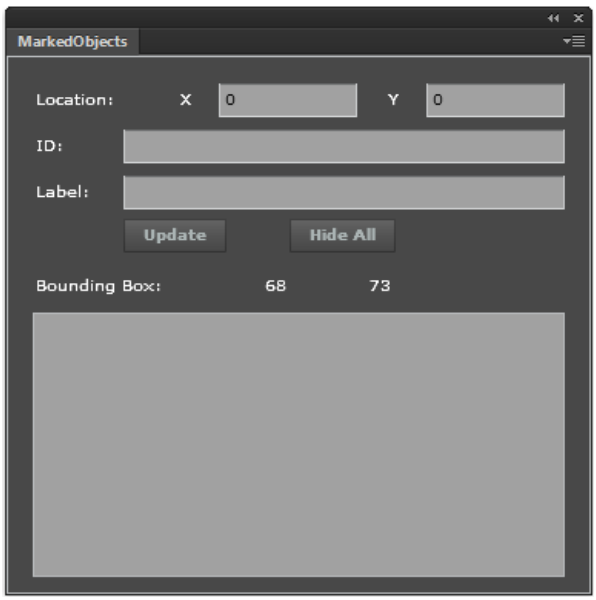
The first object has a live drop shadow. To demonstrate the "live" nature of the drop shadow, the second object shows the result of changing the shape of the object using the Filter > Distort & Transform > Pucker & Bloat tool.



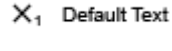
MarkedObjects

Implements a way to mark diagrams with art, in order to help lay out books during publishing. The plug-in creates the art using a tool in the Anchor group in the tool palette. When the tool is selected, you click on the document to add a new marked object. You also can view and edit the details of these marked objects from the Marked Objects dialog.

Adds the menu item Window > SDK > Marked Objects, which brings up this panel:



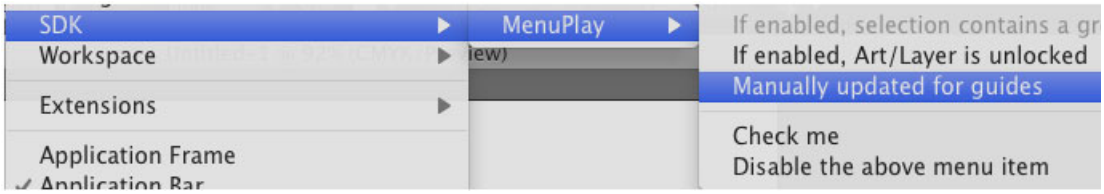
The plug-in also adds a tool in the Pen Tool group on the Tools palette, using this icon: 

This is an example of an object created by the Marked Object tool, a group containing two art items representing the cross and two text items representing the label and ID: 

MenuPlay

Demonstrates how to add and manipulate menu items.

Adds the menu item Window > SDK > Menu Play, with a submenu.



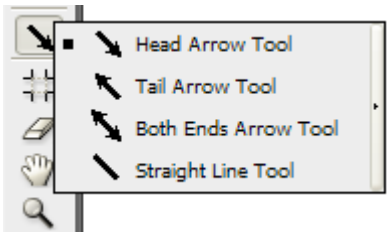
The plug-in demonstrates these types of menu manipulation:

- Highlighting menu items when particular item types are selected.
- Highlighting menu items when the art layer has a particular state.
- Changing the text displayed by the menu item when an item has a particular state.
- Allowing a menu item to affect the availability of another menu item.
- Showing the platform-specific selection marker for a toggle-type item.

MultiArrowTool

Demonstrates how to add multiple tools to the tool palette, change icons, and handle dynamic redraw of tool modifications.

This sample creates its own tool group on the tools palette, with the most recently selected arrow icon at the front of the group. It also demonstrates UI themes, by defining both a light and dark version of the tool icon.

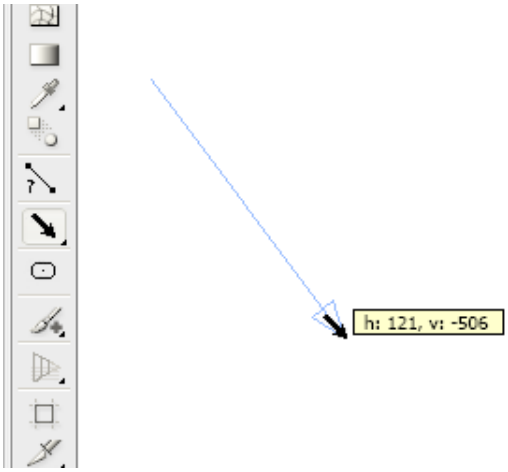


light theme

dark theme

Once one of the arrow tools is selected, the user can click and drag in the document to create a new path in the shape of an arrow. The Head Arrow tool creates an arrow with the head at the end of the path; the Tail arrow tool, at the beginning of the path.

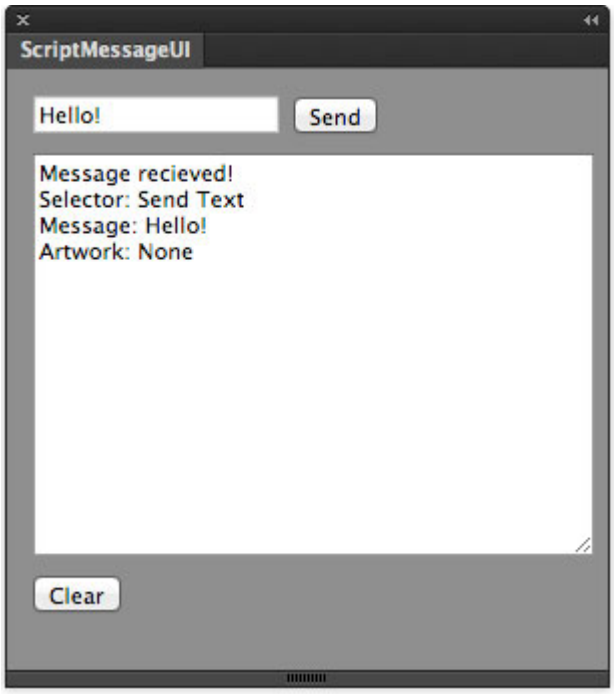
This sample demonstrates how to display an annotation while drawing a path by dragging the cursor. The annotation is activated by the left mouse button, and remains active while the button is held down.



ScriptMessage

Demonstrates how to make a plug-in scriptable using the `AIScriptMessage` structure. The sample includes a Flash extension that calls into the scriptable plug-in through ActionScript.

This sample also demonstrates the use fo the AIUIThemeSuite to detect Illustrator's UI theme and respond to theme changes by restyling the extension dynamically.



SnippetRunner

A code snippet in Illustrator is a source file, containing a class, in which you can quickly and easily prototype SDK API calls. The SnippetRunner plug-in hosts and runs code snippets within Illustrator 2019. A set of illustrative code snippets is provided. You can also use the framework to create your own snippets, enabling you to explore API without all the overhead of creating an entire plug-in.

The framework it provides includes the following:

- A user interface that allows code snippets to be chosen and run.
- A parameter-input mechanism.
- A log-output mechanism.
- Built-in unit testing of snippets, enabled through the panel fly-out menu.

This plug-in adds the menu item Window > SDK > SnippetRunner, which brings up this panel:

- Run button — Runs the selected snippet or snippet operation.
- Save log /Save log buttons — Saves or clears the log.
- Prefs button — Allows the user to specify the parameter input mode, toggle unit testing, and set the path to the assets folder. Three parameter-input modes are available:
  - Default Parameters (parameters are provided in the code)
  - Prompt Parameters (prompts are provided for parameter values)
  - CSV Parameters (values are provided in a CSV string).

Running and testing snippets

Code snippets can be run or unit tested from the operation, snippet, or category level.

The `assets/` folder contains the files used for unit testing; it also is the location where modified unit-test documents are saved. This folder is set up the first time a code snippet is run, and it remains the same until either the preference is deleted or the assets folder is changed through the Set Assets Folder option in the panel flyout menu.

The `assets/` folder must be located in `<SDK>/samplecode/CodeSnippets/examplefiles/`. This folder contains a `.ai` file and supporting image files that provide the required context for several snippet operations.

- See the `<SDK>/samplecode/CodeSnippets/` folder for the list of snippets provided with SnippetRunner, and see the brief snippet descriptions in the SnippetRunner user interface for details of what each snippet does.
- In particular, examine `SnpTemplate`, which provides a basic framework for new snippets and instructions on how to tailor the template to the new snippets needs.

Exploring the API using code snippets

The SnippetRunner plug-in makes it quick and easy to use code snippets to explore the API. It is much faster and easier to create a code snippet, rather than an entire plug-in.

All you have to do is add the template code-snippet file to the Xcode or Visual C++ project file. The template file, `<SDK>/samplecode/codesnippets/SnpTemplate.cpp`, contains the hooks you need to get any new code snippet running in SnippetRunner.

Follow the instructions in the template file to create and tailor a new code snippet to your requirements. You will have to acquire any suites your code snippet requires that are not already provided with SnippetRunner; just follow the instruction on where to add them.

StrokeFilter

This sample demonstrates how to use the AIBeautifulStrokesSuite, which was added in Illustrator CS5. The Beautiful Strokes functionality:

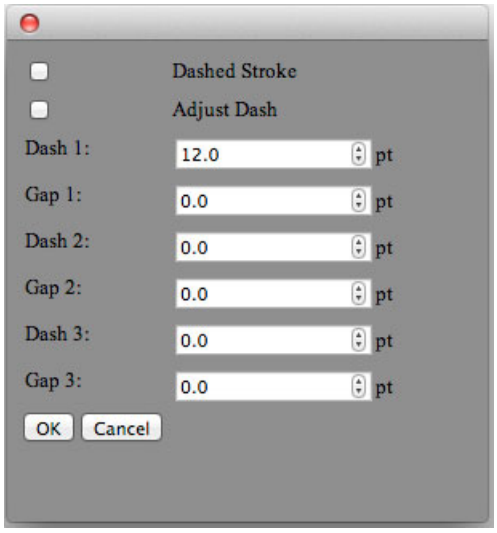
- Improves the ability to apply arrowheads.
- Enhances miter-ends behavior, changing the default miter limit from 4 to 10.
- Provides variable-width strokes, allowing you to adjust and alter stroke widths across the length of a stroke.
- Provides dash adjustment to improve control over the dashed strokes.
- Allows you to scale brushes based on path length.

This sample adds two filters, the Dashed Stroke Filter to demonstrate dash adjustment and the Waved Stroke Filter to demonstrate variable width strokes.

Dashed Strokes Filter

To invoke this filter, select a path art object, then choose Object > Filters > Additional Filters > SDK > Dashed Stroke. The resulting dialog is similar to the middle part of the Stroke Panel (Windows > Stroke):

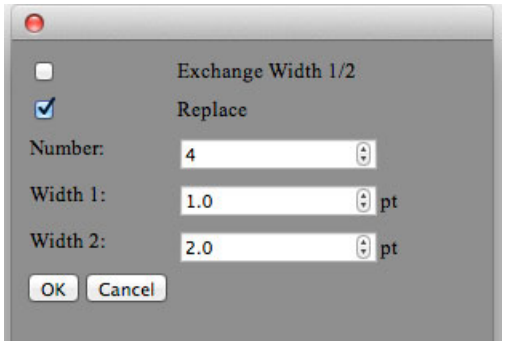
- "Dashed Stoke" controls whether the art is drawn with a dashed or solid stoke.
- "Adjust Dash" enables or disables dash adjustment.
- When the stroke is dashed, you can set a specific dashing patterns by specifying lengths for a sequence of three dashes and gaps.



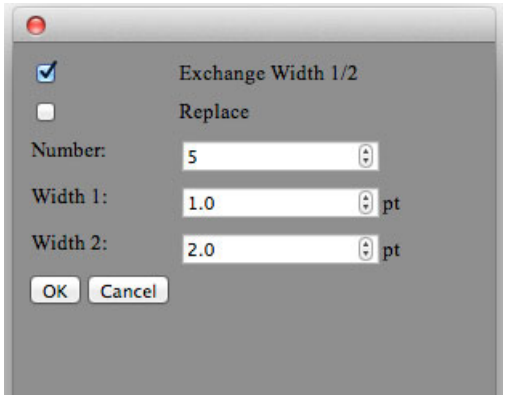
Waved Stroke Filter

To invoke this filter, select a path art object, then choose Object > Filters > Additional Filters > SDK > Waved Stroke. In the resulting dialog:

- "Number" controls the number of wave segments in the stroke.
- For *n* wave segments, variable widths are set at *n*+1 points along the path, alternating width 1 and width 2.



- Width 1 is used first, unless you select "Exchange Width 1/2".
- If you select "Replace", old width parameters are replaced by new ones. Otherwise, new width parameters are appended to old ones.



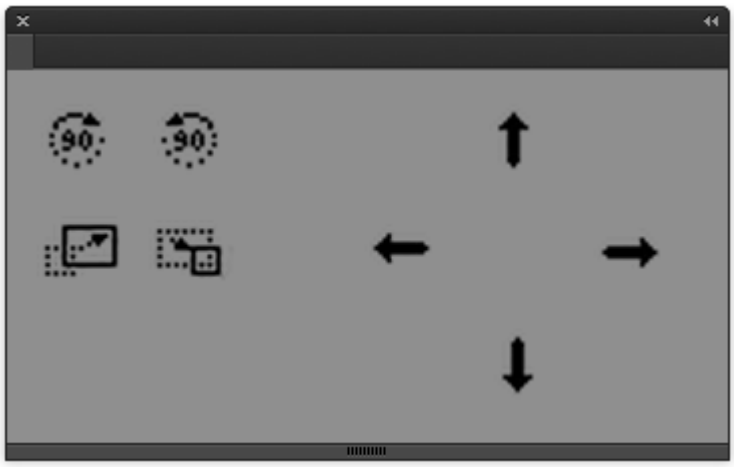
TextFileFormat

Shows how to add new file formats to Illustrator. The plug-in adds the “All Text as TEXT” format to Illustrator’s Open dialog and Save dialog. It also adds the “Selected Text as TEXT” format to Illustrator’s Save dialog.

TransformButtons

Uses the `AITransformArtSuite` suite to create a palette that executes one-click transformations to art objects.

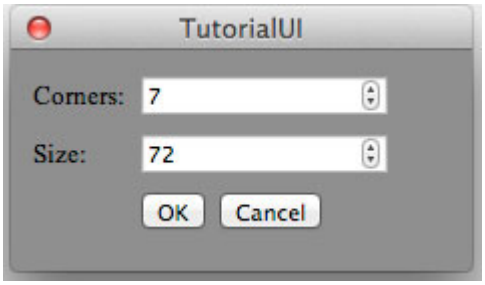
Adds the menu item Window > SDK > Transform Buttons, which brings up this panel:



Tutorial

This plug-in illustrates the basics of plug-in development. It is the sample used in the "Tutorial" section of *Adobe Illustrator 2019 Programmers Guide*.

Adds the menu item Object > Filters > SDK > Tutorial, which brings up this modal dialog:



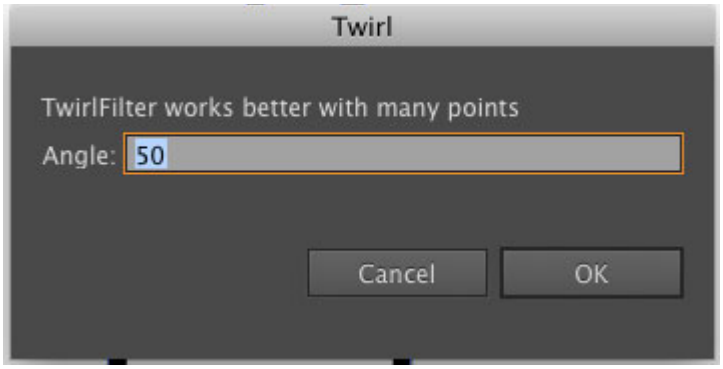
This sample creates an art item from the entered parameters. The dialog comes up with default values. The sample also defines minimum and maximum values for each data item.

This art is created using the default values:

TwirlFilter

Demonstrates how to create filters and live effects that add menu items and manipulate selected artwork. This sample adds menu items to the Filter and Effect menus, implementing the functionality of a filter and an effect.

Adds the menu items Object > Filters > SDK > Twirl., and Effect > SDK > Twirl. Both items bring up this dialog:

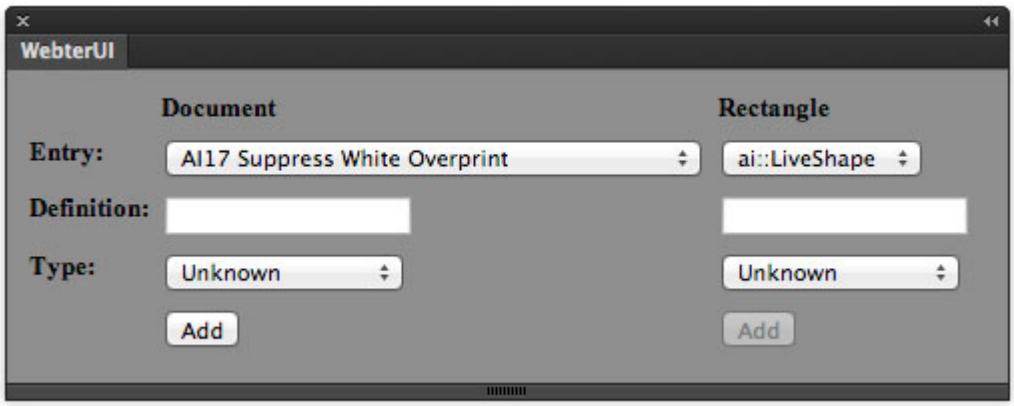


The Filter and the Effect perform the same change on the selected item. The selected item is rotated by a radian value calculated from the entered angle. The dialog comes up with a default value.

Webter

Manipulates the dictionary for document and art objects , using `AIDictionarySuite`.

Adds the menu item Window > SDK > Webter, which brings up this panel:



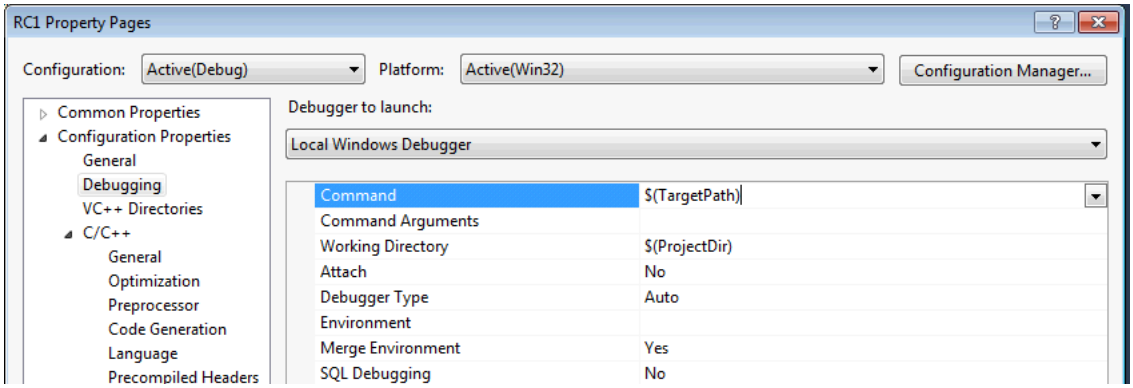
Debugging plug-ins

This section summarizes how to debug your plug-in in Visual Studio 2017 and Xcode 9.2.

Debugging a plug-in under Visual Studio

To debug your plug-in under Visual Studio:

1. Make sure your plug-in is in a location where Illustrator can find it. See "Building and running the samples" on page 9.
2. Open your plug-in project in Visual Studio.
3. Right-click the project folder and choose Properties. In the Configuration Properties, select Debugging.



Change the Command to point to your Illustrator executable. The default location is: `C:\Program Files\Adobe\Adobe Illustrator CC 2019\Support Files\Contents\Windows\Illustrator.exe`, or for 32-bit installations, under `C:\Program Files (x86)\`.

4. Set a breakpoint in your code. If you are getting started, you can set the initial breakpoint in the plug-in's entry-point function, `PluginMain()`. For plug-ins that use the plug-in framework (such as `AnnotatorPlugin.cpp`), you can set a breakpoint in `StartupPlugin()`, which is called after initialization.
5. Start the debug session using Visual Studio's Debug > Start Debugging menu.
6. Click OK to start debugging your plug-in under Illustrator.

Debugging a plug-in under Xcode

To debug your plug-in under Xcode:

1. Make sure your plug-in is in a location where Illustrator can find it. See "Building and running the samples" on page 9.
2. Open your plug-in project in Xcode.
3. Set a breakpoint in your code. If you are getting started, you can set the initial breakpoint in the plug-in's entry-point function, `PluginMain()`. For plug-ins that use the plug-in framework (such as

`AnnotatorPlugin.cpp`), you can set a breakpoint in `StartupPlugin()`, which is called after initialization.

4. Add the Illustrator application as an executable to your plug-in's Xcode project, so Xcode can find the Illustrator application. See details in "Adding Illustrator to the executable group" on page 23.

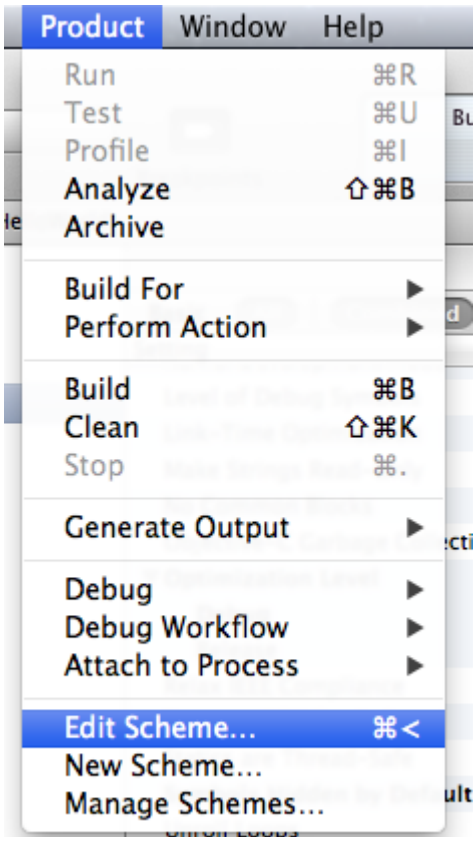
5. Start the debug session using Xcode's Debug Executable menu.

Adding Illustrator to the executable group

To debug an Illustrator plug-in from an Xcode project, Xcode must be able to start the Illustrator application. Xcode keeps this information in the Executable group for a project.

To define the executable for your project:

1. Choose Product > Edit Scheme.

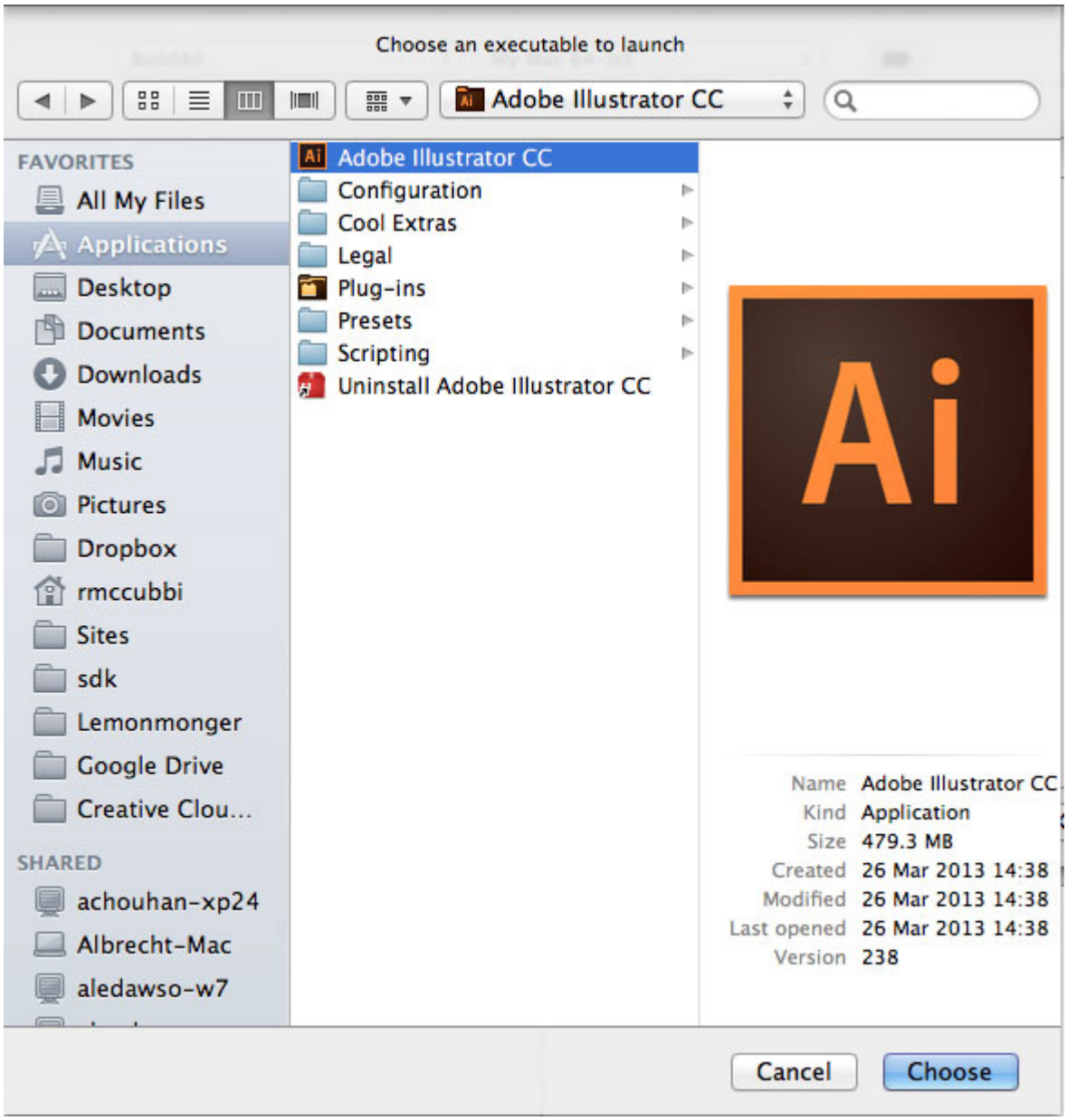


2. In the Edit Scheme dialog, click Executable and choose Other from the drop-down menu.

3. In the resulting file browser, navigate the Illustrator 2019 application. The path to the default install location is:

```
/Applications/Adobe Illustrator CC 2019/Adobe Illustrator CC 2019.app
```

4. Click Choose.



5. An Executable Info window appears, which summarizes the executable environment you just created. Click OK.

If you set up only one executable for the project, it becomes the active executable. When you choose Build > Build And Debug, Xcode starts the application and shows a debug window. You must make sure the plug-in you want to debug is loaded by Illustrator on start-up; see "Building and running the samples" on page 9.

If Illustrator is already running, you can use Run > Attach to Process in Xcode to attach the debugger and current project to the running process, as long as you have previously set the executable for the project to be Illustrator.

Setting the debug information format

Different formats are available for debugging information. Illustrator plug-ins use DWARF format (rather than Stabs format). To specify this format, set the "Debug Information Format" (`DEBUG_INFORMATION_FORMAT`) build setting.

Xcode supports two kinds of DWARF settings:

- The "DWARF" option generates symbols embedded in the `.o` object file.
- The "DWARF with dSYM File" option puts the symbols in a separate `.dSYM` file.

For more information on choosing the proper DWARF option for your plug-in, see the Xcode documentation.

Formatting program data for the debugger

The Xcode debugger variable view shows information about variable values for your program. A summary column provides additional information about a variable's content. To edit a variable summary, double-click the Summary column or choose Run > Variables View > Edit Summary Format.

When you are using an Xcode debugger to view a variable that uses a complex data structure, you may find yourself clicking to expand the class variable to see the one member you care about. To simplify, you can create a data formatter to customize how variables are displayed or summarized in the debugger. For details, see the chapter on "Examining Program Data and Information" in:

`http://developer.apple.com/documentation/DeveloperTools/Conceptual/XcodeUserGuide/Contents/Resources/en.lproj/index.html`

Problems stepping through code in the Xcode debugger

If you have problems stepping through your plug-in code using the Xcode debugger's Step Over or Step Into buttons, make sure you are using the supported version of Xcode. See "Setting up your development platform" on page 6.

If you are stepping over an Illustrator API call and you see the message, "Previous frame inner to this frame (corrupt stack?)," try one of these workarounds:

- Use the Xcode debugger's Continue button to resume execution.
- Use the Xcode debugger "Continue to Here" command to run to a subsequent line of code. The "Continue to Here" command is available by right-clicking the left of the code-editor pane in the Xcode debugger window.
- Step using gdb instead of the Step Over button. Use Xcode's Debug > Console Log menu to open the debugger console, and type *step* at the prompt.

Using the plug-in project templates

As a starting point for creating your own plug-in, we recommend you use the provided templates, which give you a new project with the same configuration as the sample plug-ins. The templates discussed in this section are in the `<SDK>/samplecode/Templates/` folder.

For detailed instructions on configuring the development environment for plug-in creation, see:

- "Creating a plug-in in Windows" on page 28
- "Creating a plug-in in Mac OS" on page 32

Visual C++ template

This template creates valid projects for both Visual Studio and Xcode. The default installation folder, indicated here by `<VS>`, is `C:\Program Files (x86)\Microsoft Visual Studio\2017`.

1. Quit Visual Studio if it is running.
2. Copy `IllustratorSDKdefault.vcxproj` to `<VS>\VC\VCWizards\`.
3. Copy the folder `<SDK>\samplecode\Templates\Win\IllustratorSDK\` to `<VS>\VC\VCWizards\AppWiz\`.
4. Copy the file `IllustratorSDK.vsz` to `<VS>\VC\vcprojects\`.
5. Create a new folder `<VS>\VC\vcprojects\IllustratorSDK\`, and copy the folder `IllustratorSDK.vsdir` into the new folder.
6. Start Visual Studio.
7. Choose File > New > Project.
8. In the Project Types pane of the New Project dialog, select Visual C++ > IllustratorPlugin.
9. In the Templates pane, select SDKProject.
10. Set the Name of the project to a name of your choice.
11. Set the Location of the project to the `<SDK>\samplecode` folder. (If you save to a different location, you will have to update the relative paths.)
12. Ensure Create Directory For Solution is not checked. (If this option is selected, the relative paths will have to be updated, as the project file will not be at the top level.)
13. Click OK.
14. Build the project and load it into Illustrator; see "Build the project" on page 31.

Xcode template

In addition to the template itself, the `Templates/Mac` folder contains a tool that you can use to create Xcode projects from the template. To use it, open a command shell and type:

```
cd <SDK_root>/samplecode/Templates/Mac
create_project.sh <ProjectName>
```

This tool follow the steps you would use to create a project from the template by hand:

1. Copy the "Template" project from `<SDK_root>/samplecode/Templates/Mac` into your `SampleCode` folder.
2. Open the Xcode Project inside your newly copied project.

3. Use the Find command to find all instances of "[PROJECT_NAME]" and replace all instances with the name of your project.

4. Rename all of the source files from "`Template*.*`" to "*`project_name`*`*.*`"
   For example, rename `TemplatePlugin.cpp` to `MyProjectPlugin.cpp`.

5. In the folders pane, expand the Targets item. Rename the "Template" bundle in that section to the name of your project (right-click "Templates" and choose Rename).

6. Double-click the project in the folders pane, then select the Build tab. Use the search box to find the string "Template", in order to narrow down the build options. Look for "Product Name" under the "Packaging" item and "PROJECTNAME" under the User-defined item. In both of these, change the value from "Template" to the name of your project.

7. Build and run Ilustrator to see your working project.

Creating a plug-in in Windows

This section describes how to write from scratch a plug-in that extends Illustrator in a very basic way. The instructions create a HelloWorld plug-in for Illustrator, which pops an alert when Illustrator starts up or shuts down.

- This section provides instructions for the Windows platform. See also "Creating a plug-in in Mac OS" on page 32.

Create a new project

1. Start Visual C++.
2. Choose File > New > Project...
3. In the New Project dialog, do the following:
   - Expand the Visual C++ filter.
   - In the Project Types window, highlight Win32.
   - In the Templates window, select Win32 Project.
   - Set the Name property of the project to HelloWorld.
   - Set the Location property of the project to `<SDK>samplecode`.
   - Deselect Create Directory For Solution.
   - Click OK.
4. In the Win 32 Application Wizard, do the following:
   - Select Application Settings.
   - Under Application Type, select DLL.
   - Under Additional Options, select Empty Project.
   - Click Finish.

Add a source file

1. In the Solution Explorer, right-click the Source Files folder.
2. Choose Add > New Item...
3. In the Add New Item dialog, do the following:
   - Expand the Visual C++ filter.
   - Highlight Code, and select C++ File (.cpp).
   - Set the Name property of the file to `HelloWorld.cpp`.
   - Click Add.

4. Copy this code to the source file, and save.

```
#include "IllustratorSDK.h"
// Tell Xcode to export the following symbols
#if defined(__GNUC__)
#pragma GCC visibility push(default)
#endif
// Plug-in entry point
extern "C" ASAPI ASErr PluginMain(char * caller, char* selector, void* message);
// Tell Xcode to return to default visibility for symbols
#if defined(__GNUC__)
#pragma GCC visibility pop
#endif

extern "C"
{
    AIUnicodeStringSuite* sAIUnicodeString = NULL;
    SPBlocksSuite* sSPBlocks = NULL;
}

extern "C" ASAPI ASErr PluginMain(char* caller, char* selector, void* message)
{
    ASErr error = kNoErr;
    SPBasicSuite* sSPBasic = ((SPMessageData*)message)->basic;
    if (sSPBasic->IsEqual(caller, kSPInterfaceCaller)) {
        AIUserSuite *sAIUser = NULL;
        error = sSPBasic->AcquireSuite(kAIUserSuite, kAIUserSuiteVersion, (const void**)
&sAIUser);
        error = sSPBasic->AcquireSuite(kAIUnicodeStringSuite, kAIUnicodeStringSuiteVersion, (const
void**) &sAIUnicodeString);
        error = sSPBasic->AcquireSuite(kSPBlocksSuite, kSPBlocksSuiteVersion, (const void**)
&sSPBlocks);
        if(sSPBasic->IsEqual(selector, kSPInterfaceStartupSelector)){
            sAIUser->MessageAlert(ai::UnicodeString("Hello World!"));
        }
    else if(sSPBasic->IsEqual(selector, kSPInterfaceShutdownSelector)){
        sAIUser->MessageAlert(ai::UnicodeString("Goodbye World!"));
    }
    error = sSPBasic->ReleaseSuite(kAIUserSuite, kAIUserSuiteVersion);
    error = sSPBasic->ReleaseSuite(kAIUnicodeStringSuite, kAIUnicodeStringSuiteVersion);
    }
return error;
}
```

5. In the Solution Explorer, right-click the Source Files folder and choose Add > Existing item.
6. Navigate to the folder `<SDK_root>\Illustratorapi\illustrator`.
7. Select the file `IAIUnicodeString.cpp`.
8. Click Add.

Add a resource file

1. In the Solution Explorer, right-click the Resource Files folder.
2. Choose Add > New Item...
3. In the Add New Item dialog, do the following:

- ▷ Expand the Visual C++ filter.
- ▷ Highlight Resource, and select Resource File (.rc).
- ▷ Set the Name property of the file to `HelloWorld.rc`.
- ▷ Click Add.

4. Close the Resource View window.
5. In the Solution Explorer, right-click `HelloWorld.rc`.
6. Select View Code.
7. Just before the line containing `#ifdef APSTUDIO_INVOKED`, insert this code:

```
#define kMySDKPluginName "HelloWorld"
#define PIPL_PLUGIN_NAME kMySDKPluginName
//////////////////////////////////////////////////////////////////////////
//
// PIPL
//
#define LC(a,b,c,d) #d, #c, #b, #a
16000 PiPL DISCARDABLE
BEGIN
    0x0001, /* Reserved (for Photoshop) */
    0L, /* kCurrentPiPLVersion */
    4L, /* Property Count */
    LC(A,D,B,E), /* vendorID */
    LC(k,i,n,d), /* propertyKey = PIKindProperty */
    0L, /* propertyID */
    4L, /* propertyLength */
    LC(S,P,E,A), /* propertyData */
    LC(A,D,B,E), /* vendorID */
    LC(i,v,r,s), /* propertyKey = PISPVersionProperty */
    0L, /* propertyID */
    4L, /* propertyLength */
    2L, /* propertyData */
    LC(A,D,B,E), /* vendorID */
    LC(w,x,8,6), /* propertyKey = PIWin32X86CodeProperty */
    0L, /* propertyID */
    12L, /* propertyLength */
    "PluginMain\0\0", /* propertyData = Entry Point */
    /* (Long Word padded C String) */
    LC(A,D,B,E), /* Vendor Key */
    LC(p,i,n,m), /* propertyKey = PIPluginNameProperty */
    0L, /* propertyID */
    12L, /* propertyLength */
    PIPL_PLUGIN_NAME "\0\0" /* propertyData = Plug-in name (padded to 4
    bytes) */
END
```

8. Save the file.

Configure the project settings

1. Right-click the project name in the Solution Explorer, and select Properties to open the Property Pages dialog.

2. In the project Property Pages dialog, expand the Configuration Properties filter.
3. Expand the C/C++ filter.
4. Choose Configuration Properties > C/C++ > General.
5. In the Additional Include Directories field, enter the following:

```
..\..\illustratorapi\ate;..\common\includes;..\..\illustratorapi\adm;
..\..\illustratorapi\illustrator\legacy;..\..\illustratorapi\illustrator;
..\..\illustratorapi\pica_sp
```

6. Choose Configuration Properties > C/C++ > Code Generation.
7. Set Runtime Library to Multi-threaded Debug (/MTd).
8. Set Struct Member Alignment to 8 Bytes (/Zp8).
9. Expand the Linker filter.
10. Choose Configuration Properties > Linker > General.
11. Set Output File to `..\output\win\debug\HelloWorld.aip`
12. Choose Configuration Properties > Linker > Input.
13. Set Module Definition File to `..\common\win\PluginMain.def`
14. Click Apply, then OK.

Build the project

1. The settings we have chosen are for a debug configuration, so ensure that Debug is selected in the drop-down list.
2. Build the project by choosing Build > Build HelloWorld.
3. Fix any build errors. See "Debugging plug-ins" on page 22.

After compiling and linking, the plug-in binary file is in the following location:

```
<SDK>\samplecode\output\win\debug\
```

You must ensure that the plug-in binary is in a location searched by Illustrator for plug-ins to load; see "Building and running the samples" on page 9.

To test the plug-in, start Illustrator 2019. During start-up, an alert is popped, displaying a short greeting. Once the application has started, your plug-in's name should be listed in the Help > System Info... dialog. During shut-down, an alert is popped, displaying another short message.

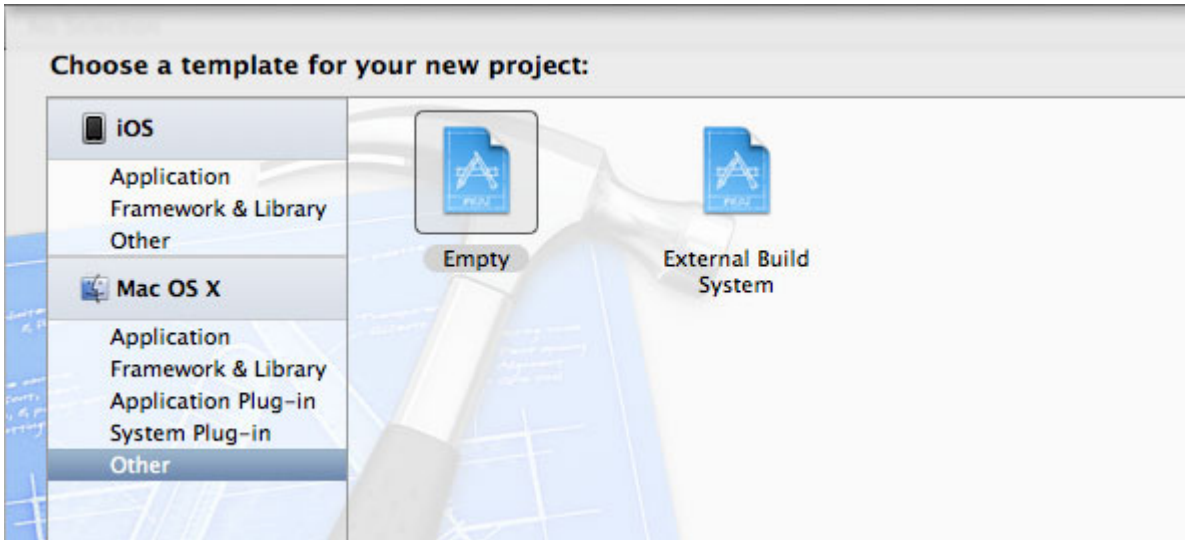
Creating a plug-in in Mac OS

This section describes how to write from scratch a plug-in that extends Illustrator in a very basic way. The instructions create a HelloWorld plug-in for Illustrator, which pops an alert when Illustrator starts up or shuts down.

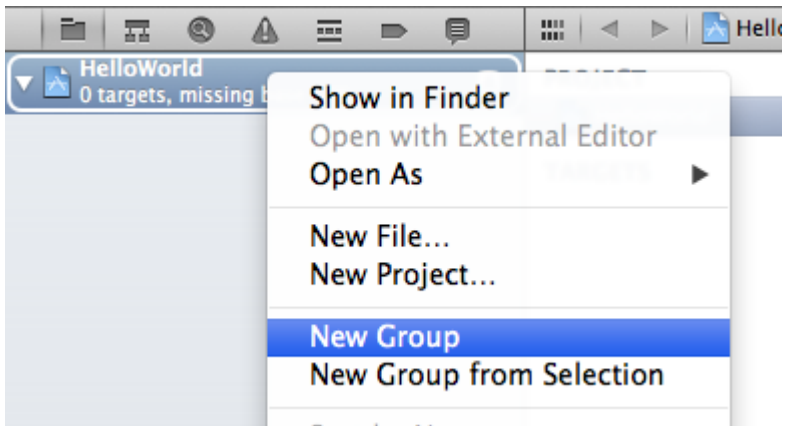
- This section provides instructions for the Mac OS platform. See also "Creating a plug-in in Windows" on page 28.

Create a new project

1. Start Xcode.
2. Choose File > New > New Project.
3. In the New Project Assistant, choose Other > Empty Project as the template and click Next.



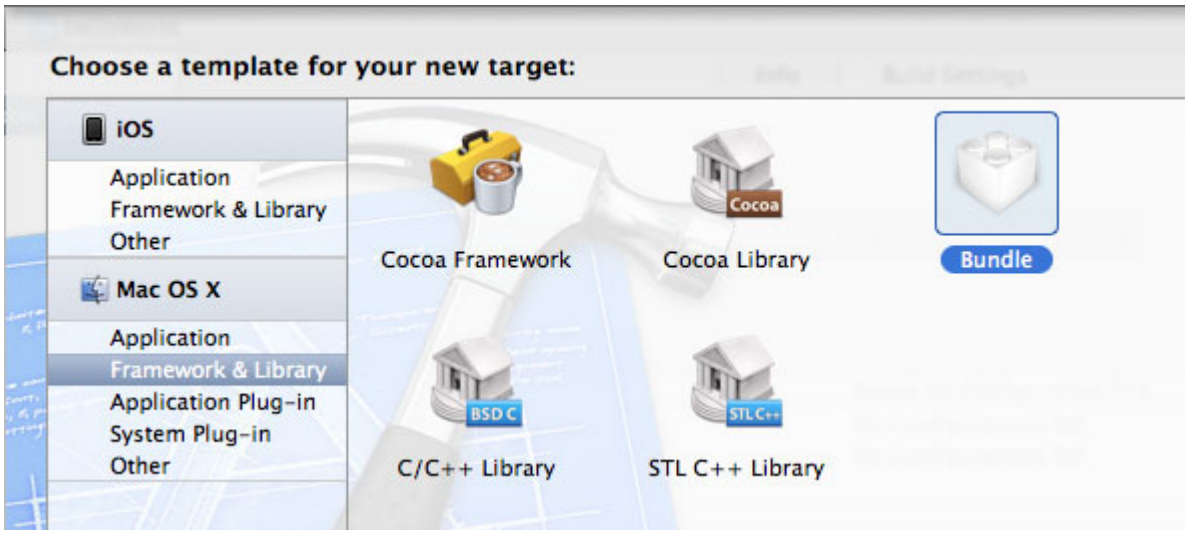
4. Set the Project Name to HelloWorld.
5. Set the Project Directory to `<SDK>/samplecode`.
6. Click Create.
7. In the main menu, choose New Group.



8. Name the new group Source.
9. Add another new group, called Resources.

Add a target

1. Click the project to show the project page, and choose Add Target.
2. In the New Target Assistant, choose Mac OS X > Framework & Library > Bundle as the template and click Next.

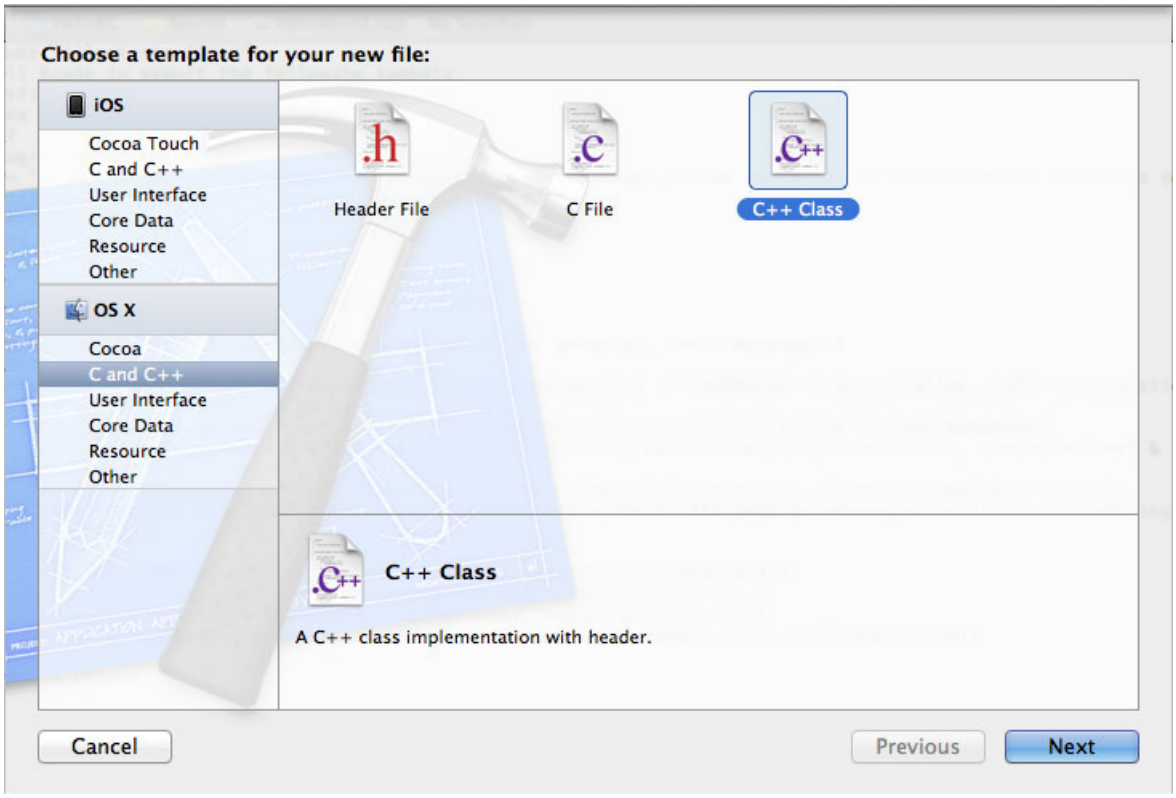


3. Set the Product Name to HelloWorld and click Finish to return to the project page.
4. In the project page, select the Build Settings tab.
5. Search for the *Wrapper Extension* setting, and set it to "aip".

Add a source file

1. RIght click the Source folder and choose New File.

2. Choose Mac OS X > C and C++ > C++ Class file as the template.



3. Name the new file HelloWorld. This creates files `HelloWorld.cpp` and `HelloWorld.h`; delete `HelloWorld.h`.

4. Open the file `HelloWorld.cpp`, copy in the following code, and save:

```
#include "IllustratorSDK.h"
// Tell Xcode to export the following symbols
#if defined(__GNUC__)
#pragma GCC visibility push(default)
#endif
// Plug-in entry point
extern "C" ASAPI ASErr PluginMain(char * caller, char* selector, void* message);
// Tell Xcode to return to default visibility for symbols
#if defined(__GNUC__)
#pragma GCC visibility pop
#endif

extern "C"
{
    AIUnicodeStringSuite* sAIUnicodeString = NULL;
    SPBlocksSuite* sSPBlocks = NULL;
}
```

```
extern "C" ASAPI ASErr PluginMain(char* caller, char* selector, void* message)
{
    ASErr error = kNoErr;
    SPBasicSuite* sSPBasic = ((SPMessageData*)message)->basic;
    if (sSPBasic->IsEqual(caller, kSPInterfaceCaller)) {
        AIUserSuite *sAIUser = NULL;
        error = sSPBasic->AcquireSuite(kAIUserSuite, kAIUserSuiteVersion, (const
void**) &sAIUser);
        error = sSPBasic->AcquireSuite(kAIUnicodeStringSuite,
kAIUnicodeStringSuiteVersion, (const void**) &sAIUnicodeString);
        error = sSPBasic->AcquireSuite(kSPBlocksSuite, kSPBlocksSuiteVersion, (const
void**) &sSPBlocks);
        if(sSPBasic->IsEqual(selector, kSPInterfaceStartupSelector)){
            sAIUser->MessageAlert(ai::UnicodeString("Hello World!"));
        }
    else if(sSPBasic->IsEqual(selector, kSPInterfaceShutdownSelector)){
        sAIUser->MessageAlert(ai::UnicodeString("Goodbye World!"));
    }
    error = sSPBasic->ReleaseSuite(kAIUserSuite, kAIUserSuiteVersion);
    error = sSPBasic->ReleaseSuite(kAIUnicodeStringSuite,
kAIUnicodeStringSuiteVersion);
    }
return error;
}
```

Add project dependencies

1. In the folders pane, right-click the Source folder and choose Add Files to "HelloWorld".
2. Navigate to the folder `<SDK_root>\Illustratorapi\illustrator`.
3. Select the file `IAIUnicodeString.cpp`.
4. Click Add.

Add a resource file

1. Under Folders, right-click the Resources folder and choose New File.
2. Under Mac OS X > Other, choose Empty as the template and click Next.
3. Set the File Name to `HelloWorld.r`.
4. Select the Target and click Create.
5. Copy the following to the new resource file:

```
#define PIPL_PLUGIN_NAME "HelloWorld"
#include "Plugin.r"
```

Configure the project settings

1. Under Folders, click the project to show the project page, and select the Build Settings tab.

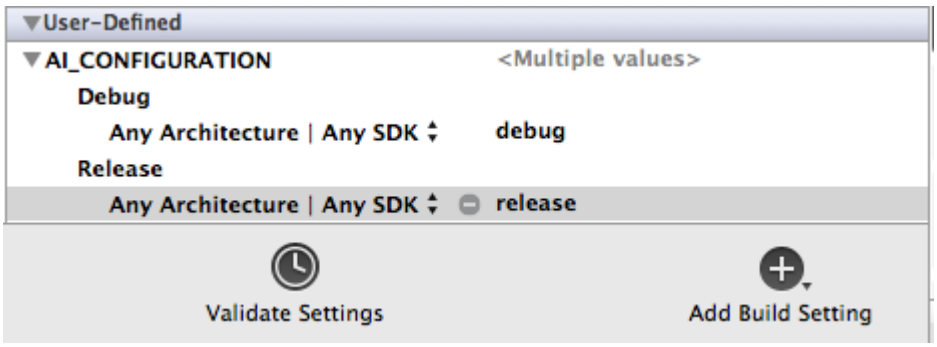
2. Set the following values:

Setting	Value
Base SDK	Mac OS X 10.12
Architecture	64-bit Intel
Compiler forC/C++/Objective C	`com.apple.compilers.llvm.clang.1_0`
Build Products Path *Per-configuration Build Products Path*	`./../output/mac/$(AI_CONFIGURATION)`
Intermediate Build Files Path	`.`

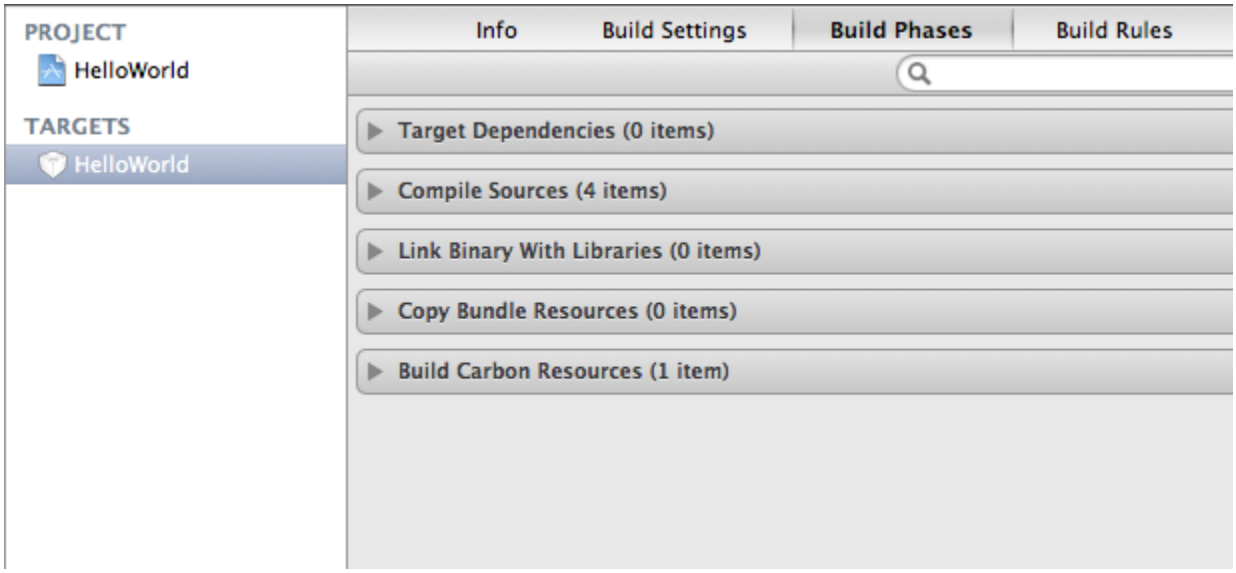
Header Search Paths	`../common/**`
	`../../illustratorapi/**`
	`/Developers/Headers/FlatCocoa/**`

Info.plist File	`../common/mac/Info.plist`
Info.plist Preprocessor Prefix File	`../common/includes/SDKDef.h`
Prefix Header > Debug	`./../common/includes/IllustratorSDKdebug.pch`
Prefix Header > Release	`./../common/includes/IllustratorSDKrelease.pch`
Deployment Target	OS X 10.12
C Language Dialect *C++ Language Dialect* *C++ Standard Library*	Compiler Default
Force Package Info Generation	Yes
Preprocess Info.plist File	Yes

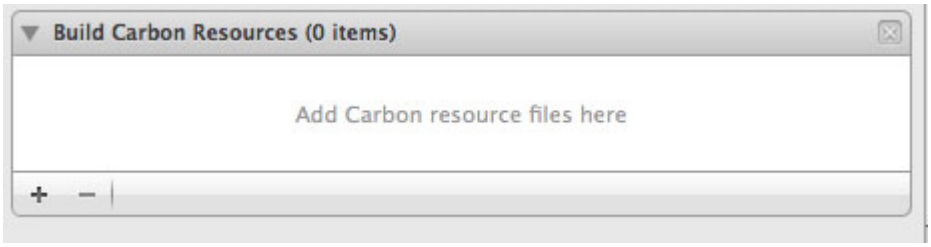
3. Choose Add Build Setting > Add User-Defined Setting, and enter a setting named "AI_CONFIGURATION."
   Expand its definition and set the Debug value to "debug", and the Release value to "release":



4. Under Targets on the left, select the HelloWorld target and navigate to the Build Phases tab.



5. Click Add Build Phase and choose "Add Build Carbon Resources".



6. Click "+" and choose `HelloWorld.r` as the new resource.

Build the project

1. Choose Product > Edit Scheme, and select "Debug" as the build configuration.
2. Build the project by choosing Product > Build.
3. Fix any build errors. See "Debugging plug-ins" on page 22.

After compiling and linking, the plug-in binary file is in the following location:

```
<SDK>/samplecode/output/mac/debug/
```

You must ensure that the plug-in binary is in a location searched by Illustrator for plug-ins to load; see "Building and running the samples" on page 9. To test the plug-in, start Illustrator 2019. During start-up, an alert is popped, displaying a short greeting. Once the application has started, your plug-in's name should be listed in the Help > System Info... dialog. During shut-down, an alert is popped, displaying another short message.