ADOBE® ILLUSTRATOR® CS3

ADOBE ILLUSTRATOR CS3 PORTING GUIDE

Adobe®

Adobe Illustrator CS3 Porting Guide

Technical Note #10500

Contents

Adobe Illustrator CS3 Porting Guide

This document describes how to update your SDK plug-in code and development environments for Adobe® Illustrator® CS3. It details changes in the public API and other aspects of the SDK since the previous release.

Introduction

Before you start porting your Illustrator CS2 SDK projects and code to Illustrator CS3, read this introduction, which provides an overview of the document content and conventions.

To begin porting:

1. Check that your system meets the basic requirements specified in "Development Platforms" on page 6.
2. Install the SDK.
3. Set up your machine for development by following the instructions in "Visual C++ 2005 Transition" on page 7 and "Xcode Transition" on page 11.
4. Examine the documentation, then compile and run the samples.
5. Follow the recommended procedures to port your own plug-in projects and code.

In this Document

This document contains the following sections:

- "Visual C++ 2005 Transition" on page 7 — Project changes and update procedures for the Windows® platform.
- "Xcode Transition" on page 11 — Project changes and update procedures for the Macintosh® platform.
- "SDK Changes" on page 35 — A summary of changes to the organization of the SDK since the previous release.
- "Illustrator API Changes" on page 40 — A summary of important changes to the API since the previous release.
- "Porting Case Study" on page 46 — Step-by-step description of the changes you need to make to port one of the samples from the Illustrator CS2 SDK.

Terminology

The following terms are used in this document:

- *API* — Application programming interface.
- *Application* — Illustrator CS3, unless otherwise specified.
- *SDK* — Software development kit for the application.

Notational Conventions

- SDK root folder — <*SDK*> is used to indicate your locally installed SDK root folder. The actual root location depends on the installation and operating system.
- Menu names — The right angle bracket, >, is used to indicate menu hierarchies. For example, Tools > Options refers to the Options menu item on the Tools menu.
- Computer input or output (including source code) is shown in monospace font; for example:

```
Cannot find specified file
```

Development Platforms

Supported platforms for Illustrator plug-in development are shown in Table 1.

TABLE 1 Supported Development Platforms

Platform	Component	Note
Mac OS® 10.4	Mac OS 10.4 or higher	Mac OS 10.3 and earlier are not supported.
	Illustrator CS3	
	Xcode 2.4.1	Xcode can be downloaded from http://developer.apple.com/tools/download/.
Windows XP	Service Pack 2 or higher	
	Illustrator CS3	
	Visual C++ 2005	Also known as Visual C++ 8, a component of Visual Studio 2005.

You also need the components shown in Table 2 to work with the CS3 SDK:

TABLE 2 Additional Required Components

Component	Note
Adobe Reader 6.0 or later	To view PDF-based documentation.
CHM viewer	To view compiled HTML documentation (index.chm) in Mac OS, you must use a CHM viewer for Mac OS. Two such viewers are chmox (http://chmox.sourceforge.net/) and xCHM (http://sourceforge.net/projects/xchm/).

Visual C++ 2005 Transition

Visual Studio Set-up

Developing plug-ins for Illustrator CS3 requires Microsoft® Visual C++ 2005 (also known as Visual C++ 8), which is a component of Microsoft Visual Studio 2005. If you have the Professional or Standard Editions of Visual Studio 2005, you must install the Visual C++ components.

Project Conversion

The new project file format is quite similar to the Visual Studio .NET 2003 version, so this is a very simple move. Visual Studio does most of the work.

Initial Conversion

Start with a clean tree. This means you should refresh your source tree in such a way that no compiler-generated files are present, including .sln and .obj files. Open the old .vcproj file with Visual Studio 2005, and let it convert the project. Choose whether to make a backup copy, and proceed. (A backup copy may not be needed if you already have the file under revision control.)

Conversion Log

When the conversion finishes, the conversion report shows some warnings about the secure version of the C Runtime Library. Microsoft has introduced their own, proprietary versions of C run-time functions that remove various vulnerabilities in the C run-time library. These are platform specific and not part of Illustrator. Use of this library is turned on by default. You may choose to incorporate these changes, but—if you also are building for Mac OS—it will be up to you to provide cross-platform support. *If you leave your code as is, be aware that your Windows code is less secure.*

The conversion report also shows some details of changes to the compiler that may require you to alter your code. Handle these case by case. The product and SDK sample code were updated for these changes as needed.

Configuring the Project Settings

After converting the project, make sure you are using the right settings. Follow these steps:

1. Choose Project > Properties.
2. From Property Pages, choose Configuration Properties > C/C++ > General.
3. Set Debug Information Format to Program Database (/Zi).
4. Select the Code Generation tab.
5. Set the Runtime Library property to Multi-threaded Debug (/MTd).
6. Ensure that the Struct Member Alignment is set to 4 Bytes (/Zp4).
7. Select the Advanced tab.
8. Set Compile As to Compile As C++ Code (/TP).

Adding Extra include Paths

Your existing include paths do not cover the ATE files, which have moved; see "Adobe Text Engine API Files Moved" on page 36. Add <*SDK*>/illustratorapi/ate/ to the additional include path.

For details on files that moved and instructions on updating your project, see "SDK Changes" on page 35.

Building the Project

After completing these updates, you can build the project. If you have been using Visual C++ .NET 2003, Visual C++ 2005 should have a familiar feel:

- The Build command is still in the Build menu. You may find it uses more memory.
- Starting the debugger is slower, especially if you are starting it from a project with multiple projects in one solution file. We recommend you start the debugger from a solution containing one project.

Dealing with Visual Studio Warnings

You may see warnings concerning deprecated methods from the windows API, like sprintf and strncpy. Visual Studio will suggest an appropriate replacement to use. The SDK incorporates these method name changes in IllustratorSDK.h in <*SDK*>/samplecode/common/includes. All other warnings encountered when porting the SDK projects are documented in "Known Visual C++ 2005 Issues" on page 9.

Known Visual C++ 2005 Issues

The following issues were found when porting SDK projects to Visual C++ 8.

warning C4290: C++ exception specification ignored except to indicate a function is not declspec(nothrow)

When compiling any project which includes ASMemory.cpp, the Visual C++ 8 compiler may give this warning.

The function in question is declared using exception specification, which Visual Studio accepts but does not implement. Code with exception specifications that are ignored during compilation may need to be recompiled and linked to be re-used in future versions supporting exception specifications. The following line disables this warning:

```
#pragma warning (disable: 4290)
```

The <*SDK*>/samplecode/common/includes/pragma.h file contains this line. Include this file in your top-level header file or add it to the project as a forced include by typing the filename into the Force Includes setting in Property Pages > Configuration Properties > C/C++ > Advanced. If the file is not in an included path, use the entire relative path.

warning C4103: "<file path>" alignment changed after including header, may be due to missing #pragma pack(pop)

The Visual C++ 8 compiler gives this warning if your project's struct member alignment setting conflicts with the 4-byte alignment required by the API.

Header files ADMHeaderBegin, ADMHeaderEnd, AIHeaderBegin, and AIHeaderEnd are now included by the appropriate API headers, to enforce a struct member alignment of 4 bytes. If your project requires a different struct member alignment for non-API structures, you can ignore the warning. Otherwise, we recommend you set the default alignment in your project to match that used by the API; see Step 6 in "Configuring the Project Settings" on page 8.

warning C4800: "AIBool8" : forcing value to bool "true" or "false" (performance warning)

When compiling any project that includes the helper class APIs (like IAIFilePath.cpp), the Visual C++ 8 compiler may give this warning. The warning can be ignored and has been disabled for sample code (see <*SDK*>/samplecode/common/includes/pragma.h). Use of the bool type was removed from all suite APIs, to preserve binary compatibility; however, the helper class APIs still use bool. See "Removal of bool Types from Suite APIs" on page 45.

Microsoft Run-time Assert Dialog

When debugging, be aware that Visual C++ 8 has much stronger iterator checking at run time than Visual C++ 7, CodeWarrior, or Xcode. It refuses to allow things the other environments allow but are technically illegal. If you see a Microsoft run-time assert dialog during execution that looks different than the standard assert dialog, with a pointer/STL assertion and many variables prefixed by underscores, click the button to go into the debugger. If the assertion was

fired from inside their STL code (look up the call stack), look at the last place in your code to see what was happening. Typically, you will find some abuse of an iterator or STL container.

Stronger iterator checking is helpful for finding problems before they occur. Consider the following example:

```
vector<int> v;
...
vector<int>::iterator i = v.begin();
v.push_back (5);
int x = *i;
```

This code is wrong. Unless v.capacity() is greater than v.size(), the call to push_back will reallocate the vector's memory to make room for inserting the new element. If v reallocates, the iterator i is no longer valid.

Without iterator checking, you can find bad code only by encountering a case at run time, in which some misbehavior was caused by an invalid iterator.

With full iterator debugging, v marks all iterators as invalid on push_back. Subsequent de-references always cause an assert, regardless of whether v actually reallocates. This enables you to always find a class of problems that occur sporadically.

Here are some other common iterator mistakes:

- Decrementing an iterator that is already at container.begin().
- Incrementing an iterator that is already at container.end().
- De-referencing an invalid iterator (like container.end()).
- Going beyond vector bounds (like myVector[-1]).
- Trying to find the address of the start of an empty vector's storage (for example, calling &v[0] when v.empty() is true).

fatal error C1083: Cannot open include file: "ATETypesDef.h": No such file or directory

API files for the Adobe Text Engine (ATE) have moved to a different folder; see "Adobe Text Engine API Files Moved" on page 36. Update your Visual C++ 8 project to add <*SDK*>/illustratorapi/ate/ to the include paths folder.

SDK projects use these paths:

```
.\Source,.\Resources,..\common\includes,..\..\illustratorapi\adm,..\..\illustratora
pi\ate,..\..\illustratorapi\illustrator,..\..\illustratorapi\illustrator\actions,..
\..\illustratorapi\pica_sp,..\..\illustratorapi\illustrator\legacy
```

fatal error C1083: Cannot open source file: '..\..\illustratorapi\illustrator\IText.cpp': No such file or directory

This is reported for any ATE file used in your project from the old ATE location. Frequently used files are IText.cpp and IThrowException.cpp.

fatal error C1010: unexpected end of file while looking for precompiled header. Did you forget to add '#include "IllustratorSDK.h" to your source?

This error is reported when an API helper class is added to a Visual Studio project and you forget to turn off use of pre-compiled headers for that file. All sample projects use the SDK's pre-compiled headers, but the API helper classes do not.

For example, the error text displayed in the title was caused by the addition of IAIUnicodeString.cpp to an SDK sample project. All sample projects on Windows use IllustratorSDK.h as a pre-compiled header, and any file added to an SDK project automatically inherits these pre-compilation properties. The error occurs because Visual Studio expects the source file to include the pre-compiled header IllustratorSDK.h; however, the helper-class source files provided by the API, such as IAIUnicodeString.cpp, do not include this file.

To fix this error, configure the Use Precompiled Header property for that file in the project settings. Using IAIUnicodeString.cpp as an example, follow these steps:

1. Add IAIUnicodeString.cpp to the project.
2. Right-click on IAIUnicodeString.cpp, and select Properties.
3. Choose Configuration Properties > C/C++ > Precompiled Headers.
4. Set Create/Use Precompiled Header to Not Using Precompiled Headers.

Xcode Transition

On 6 June 2005, Apple® announced at its Worldwide Developers Conference that it planned to deliver models of its Macintosh computers using Intel microprocessors by June 2006 and transition all its Macintosh computers to Intel microprocessors by the end of 2007. For more information about the announcement, see the press release at http://www.apple.com/pr/library/2005/jun/06intel.html. Because there will be Macintosh computers with Intel and PowerPC processors capable of running Mac OS X in the foreseeable future, Apple recommends developers build their Mac OS X applications as universal binaries.

Universal binaries run natively on Macintosh computers using PowerPC or Intel microprocessors and deliver optimal performance for both architectures in one package. The Apple Xcode development tool provides the most convenient way to produce universal binaries; therefore, Adobe decided to switch to Xcode from CodeWarrior for the Illustrator CS3 development environment.

An Xcode template for Illustrator SDK projects is included in this SDK. This template sets up a skeleton Xcode project for an Illustrator SDK plug-in and is the recommended approach to use when porting your plug-in project from CodeWarrior.

Xcode Set-up

Install Mac OS 10.4

The supported version of Xcode, version 2.4.1, requires version 10.4 of the operating system also known as Tiger. From the Tiger DVD, the easiest install option is upgrading your current system. If you reboot from the DVD, this happens automatically and overwrites your 10.3 operating system with new components from version 10.4. From most of the user experiences we have heard so far, this option works for Illustrator CS3 development.

If you want to start with a fresh system, use Archive Install. This leaves your home folder but moves all system and common files to a "Previous Systems" folder on your hard disk. This breaks the Adobe licensing for Suite products, unless you copy the correct items back into /Library/Application Support/Adobe/.

The most drastic approach is to back up and erase your hard disk and install a completely fresh system.

Install Xcode 2.4.1

Before installing Xcode 2.4.1, you must cleanly uninstall any previously installed version of Xcode. For instructions on how to uninstall, see the Read Me file in the Xcode package. If you do not already have a copy of Xcode 2.4.1, download it from the Web site listed in Table 1.

Anatomy of an Illustrator SDK Xcode Project

This section introduces the key components of a typical Illustrator SDK Xcode project, describing how an Illustrator SDK project is structured using Xcode. The Xcode documentation provides detailed information about Xcode.

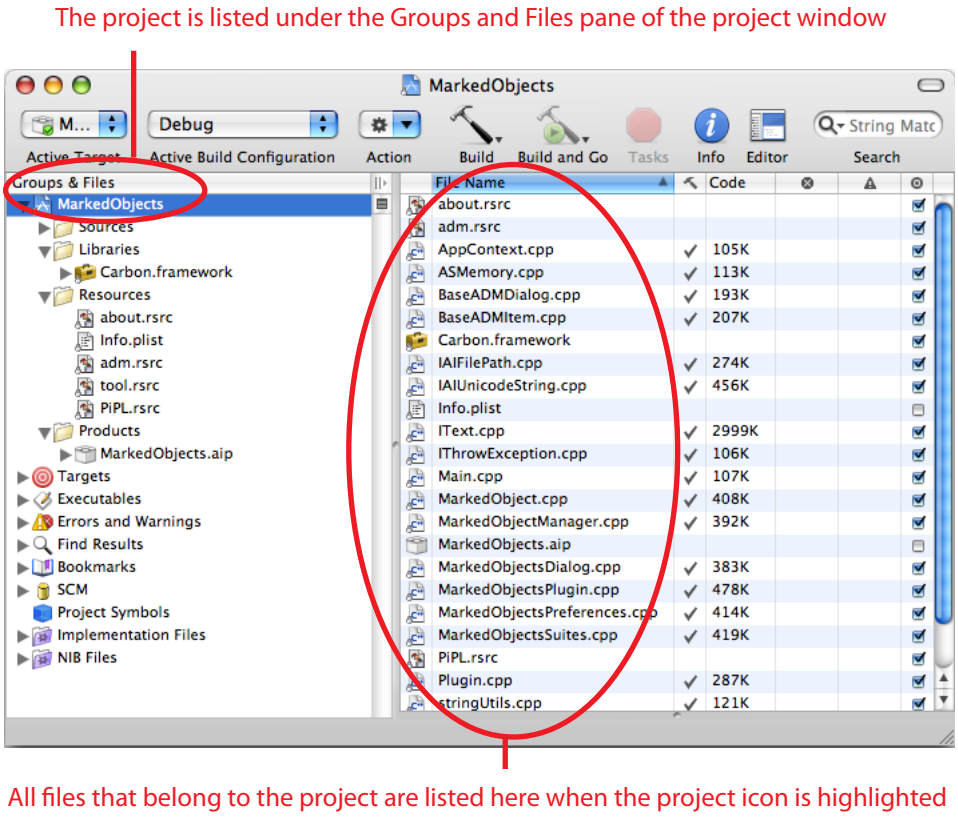
NOTE: This document assumes the default layout view in Xcode.

Project Group

The project group provides access to all files belonging to a project. It is represented by the blue project icon at the top of the Groups & Files list. The project group organizes all project files into groups, represented by the yellow folder icons under the project icon. As shown in Figure 1, an SDK project group has these components:

- Sources — Contains the C++ source files for the projects.
- Libraries — Contains Carbon.framework, which is required by every Illustrator SDK project.
- Resources — Contains the resource files for the projects, including .plist and .rsrc files.
- Products — Contains the product that will be built by the project, identified with the framework name specified in the target's Product Name setting.

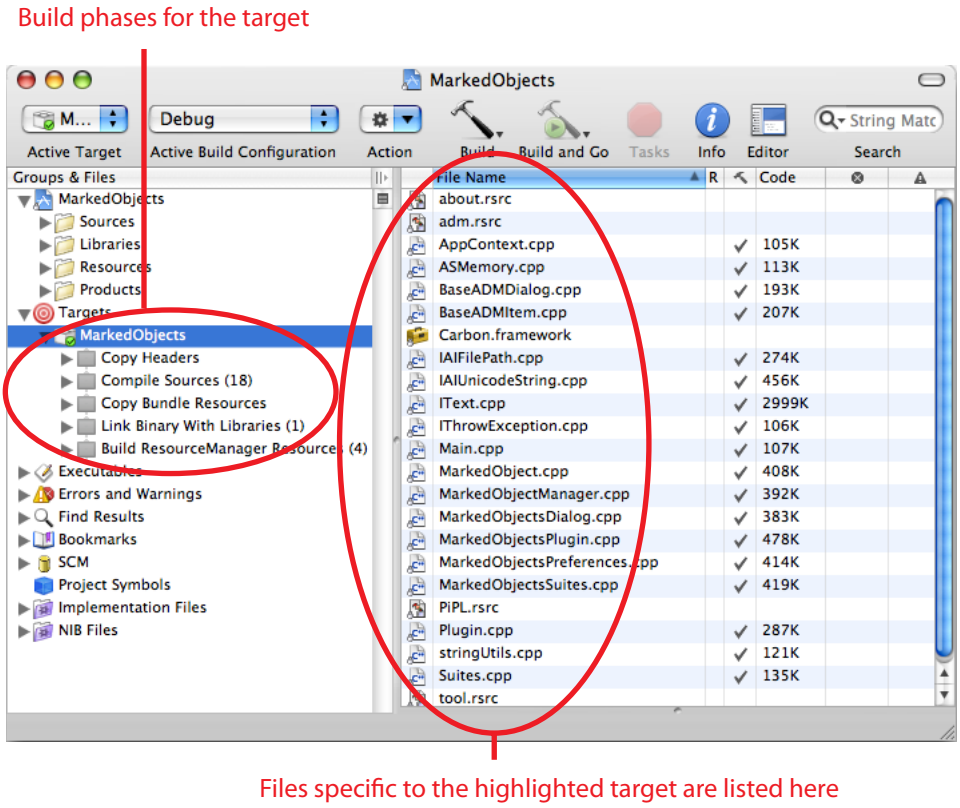
FIGURE 1 ***A Typical SDK Project's Project Group***



Target Group

A target contains the instructions for building the finished product from a set of files in your project. You may see many Apple Xcode sample projects using targets to build different products within the project. Each product can have different configurations that developers can use to configure how their product is built during different phases of development. For a short description of each build phase, see Table 3. For a typical target group, see Figure 2.

FIGURE 2 ***A Typical SDK Project's Target Group***



An Illustrator SDK Xcode project target contains two main components that provide the inputs to Xcode that are required to build an Illustrator plug-in: *files* and *build phases*.

Files

Choose Groups & Files > Targets and select the project target; the right side of the project window shows all files added to the target. Files are added to a target when they are added to the project; for a file to show the path to the file, see "Adding Existing Source Files to the Project" on page 22. Choose Get Info.

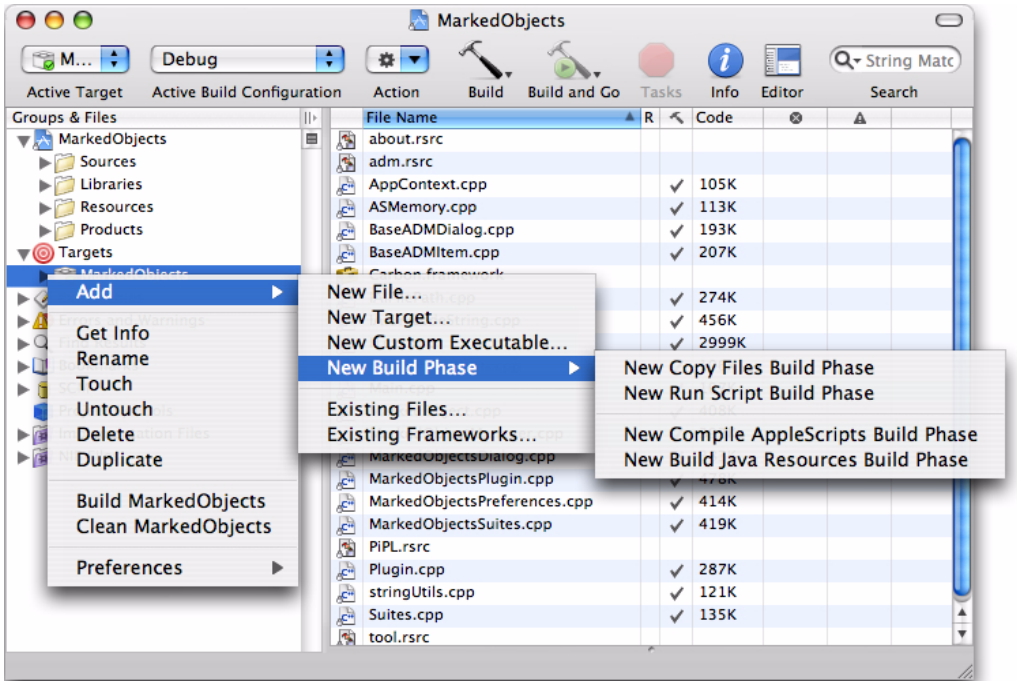
There is only one target in an Illustrator SDK project; it has a debug and release configuration, which specify different settings. If your debug and release configurations require different files or different versions of files, you can add a target, using one to build the debug target and the other to build the release target. If there are two targets, highlighting each target shows which files were added to that target.

Build Phases

In Xcode, a build phase represents a task to be performed on a set of files, like compiling a C++ source file using GCC. To add a build phase, follow these steps:

1. Right-click the target in Groups & Files > Targets to bring up the context menu, as shown in Figure 3.
2. Select a build phase item from the Add > New Build Phase menu.

FIGURE 3 Adding a Build Phase



An Illustrator SDK Xcode project contains five build phases, as shown in Table 3.

TABLE 3 Illustrator SDK Xcode Project Build Phases

Build Phase	Description
Copy Headers	Copies header files with "public" or "private" roles into the product's Headers folder.
Compile sources	Compiles source files according to the build rules. This includes compiling the C++ files using GCC 4.0, and any resource files that need to be compiled.
Copy bundle resources	Copies resource files from the project's temporary folder into the product's Resources folder.
Link binary with libraries	Links object files generated from the previous phase against frameworks and static libraries to produce a binary file. In general, a standard SDK project links against Carbon.framework.
Build ResourceManager resources	Creates the final resource file using all the individual .rsrc files included for this phase.

Executables Group

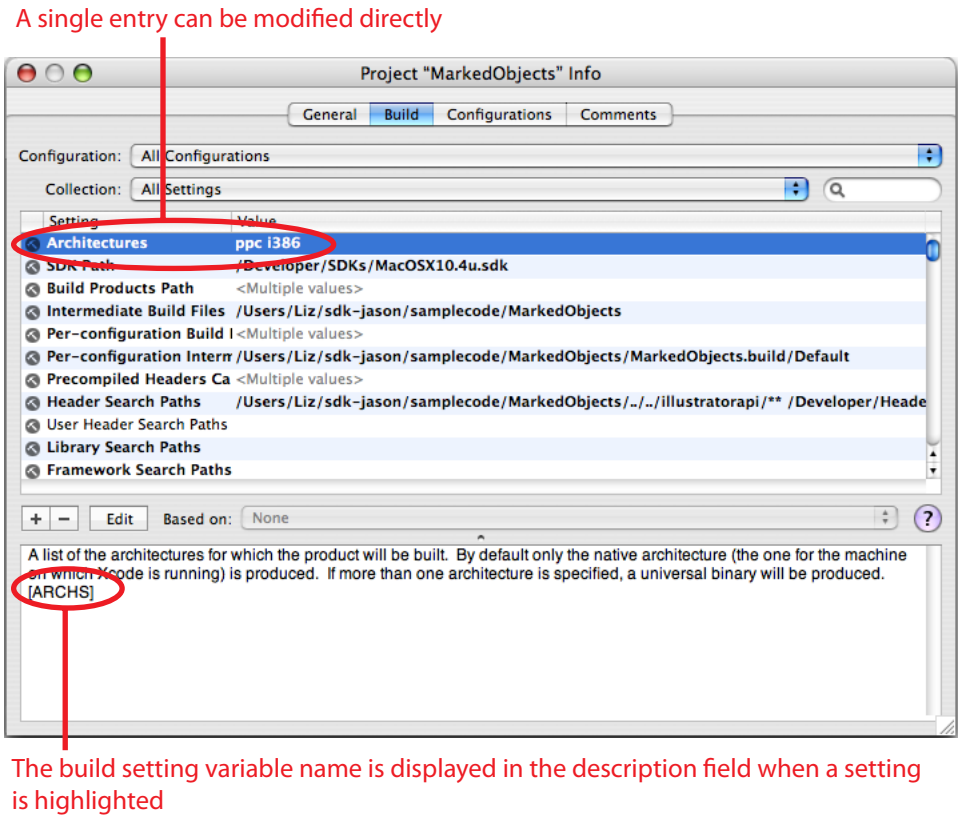
There is no default executable defined for an Illustrator SDK Xcode project, but to debug your plug-in, you must add one. For details, see "Debugging a Plug-in under Xcode" on page 32.

Build Settings

A build setting in Xcode is a variable that specifies a required project-build value, like the architecture for which the product should be built. It also can specify project properties, like product name, so the build system can package the product using the name of your choice. Different build commands have different kinds of command options, and you can use a build setting to specify what option a specific command should use.

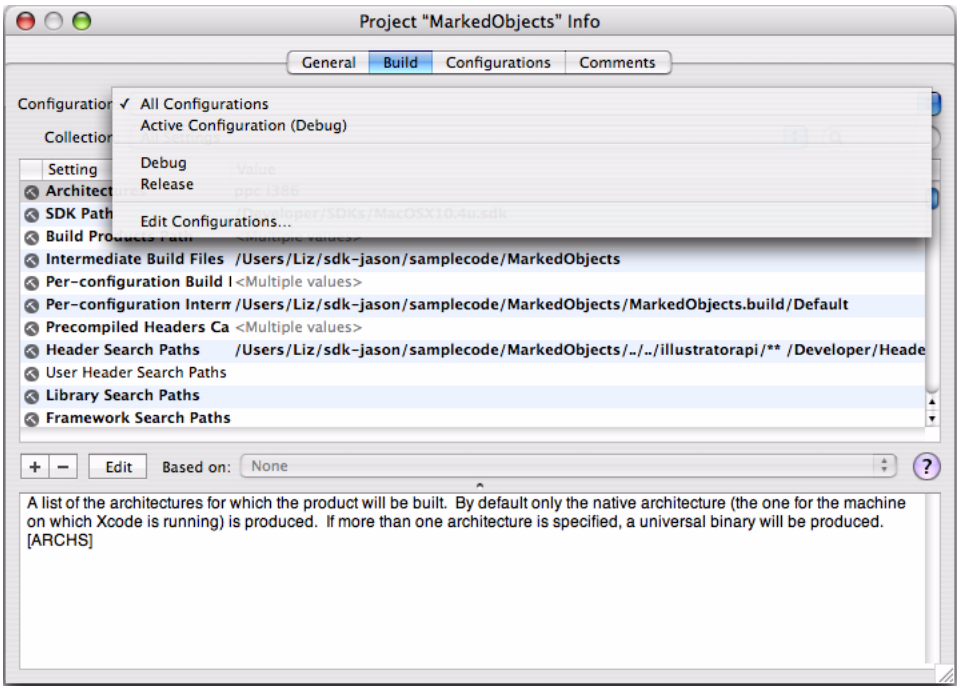
Figure 4 shows some project-level build settings of an Illustrator SDK project. Each setting is represented by a variable name, which is displayed in the brackets at the end of the setting description when the setting is highlighted. To use a setting variable in a command, use the syntax $(VARNAME). For example, the setting that defines what architecture for which the product should be built is represented by the variable ARCHS, and you use $(ARCHS) to pass it into the linker command.

FIGURE 4 ***Illustrator SDK Project Settings***



Illustrator SDK Xcode projects use debug and release build configurations to control the different build settings required for the debug and release versions of the product. The project build settings are controlled through the project Get Info dialog, as shown in Figure 4 and Figure 5.

FIGURE 5 ***The Configuration drop down box***



The project-level settings override the default Xcode project settings (that is, the settings with which a new project is created automatically). Any target-level settings override the project-level settings; however, Illustrator SDK projects do not manipulate the settings at the target level, because there only is one target.

Before making changes to any settings, it is important to check which configurations the changes will be applied to: all, debug or release. Choose All Configurations to edit any settings relevant to both debug and release configurations. Select Debug to edit debug-specific settings, and Release to edit release-specific settings. See Figure 5.

To see which settings are specific to a configuration or common to all configurations, open the project-level Get Info dialog and select the Build tab. Settings applicable to all configurations are visible when All Configurations is selected, while those specific to a particular configuration are visible only when that configuration is selected. If the configurations differ, <Multiple Values> is displayed next to the project setting.

Creating an Xcode Project for an Illustrator SDK Plug-in

An Xcode template for Illustrator SDK plug-in projects is included in the SDK. The template sets up project settings and adds the appropriate resources and the required framework for the target. You can use the template to set up an Xcode project for an existing plug-in or create a new plug-in project for Xcode.

Setting up the Xcode Project Template for an Illustrator SDK Plug-in

Use the IllustratorPlugin template to convert or create a new Xcode project for an Illustrator plug-in. The template is provided with the SDK in the following folder:

`<SDK>/samplecode/Templates/Mac/IllustratorPlugin/`

To make the template available to Xcode, copy the IllustratorPlugin folder to the following folder on your start-up disk volume:

`/Library/Application Support/Apple/Developer Tools/Project Templates/`

Creating an Illustrator SDK Xcode Project from the Template

You can use the template to generate an empty Illustrator SDK project for Xcode, using these steps:

1. Choose Xcode > File > New Project to bring up the New Project Assistant window. Scroll down the window to find a group called IllustratorPlugin. Expand the group to show SDK-Project, as shown in Figure 6.

NOTE: If you do not see the IllustratorPlugin template, check that you copied the template to the correct location; see "Setting up the Xcode Project Template for an Illustrator SDK Plug-in" on page 19.

2. Select SDKProject and click Next in the Assistant window, to bring up the New SDKProject window, as shown in Figure 7.

3. Give the project a name (preferably without spaces; see "Updating Relative Paths" on page 23), and set the Project Directory to the SDK's samplecode folder:

   `<SDK>/samplecode/<projectname>`

4. Click Finish.

FIGURE 6 ***Illustrator Project Template for Xcode***

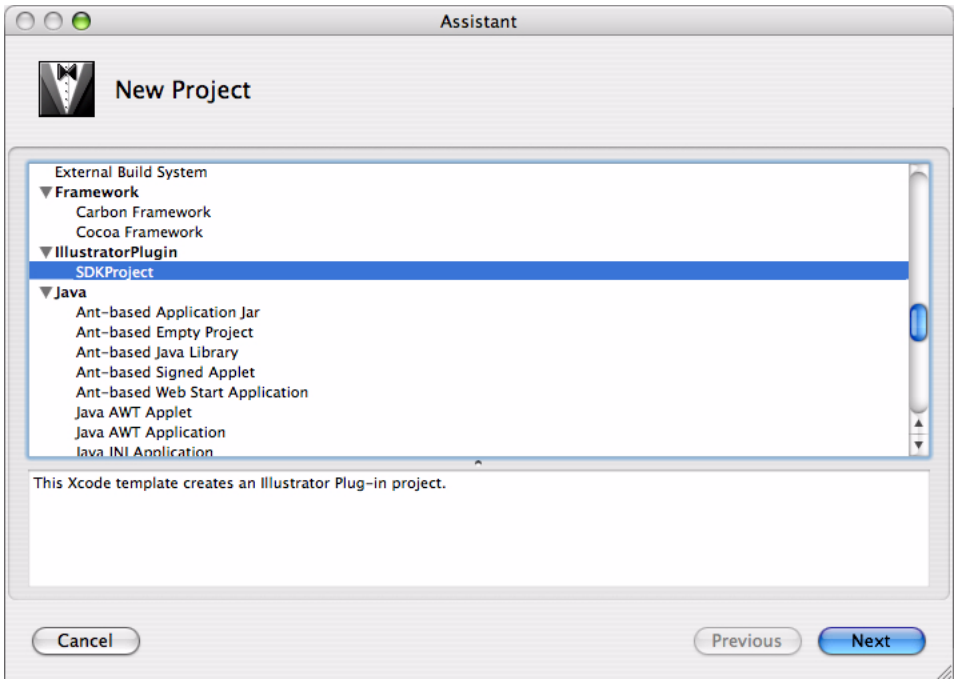
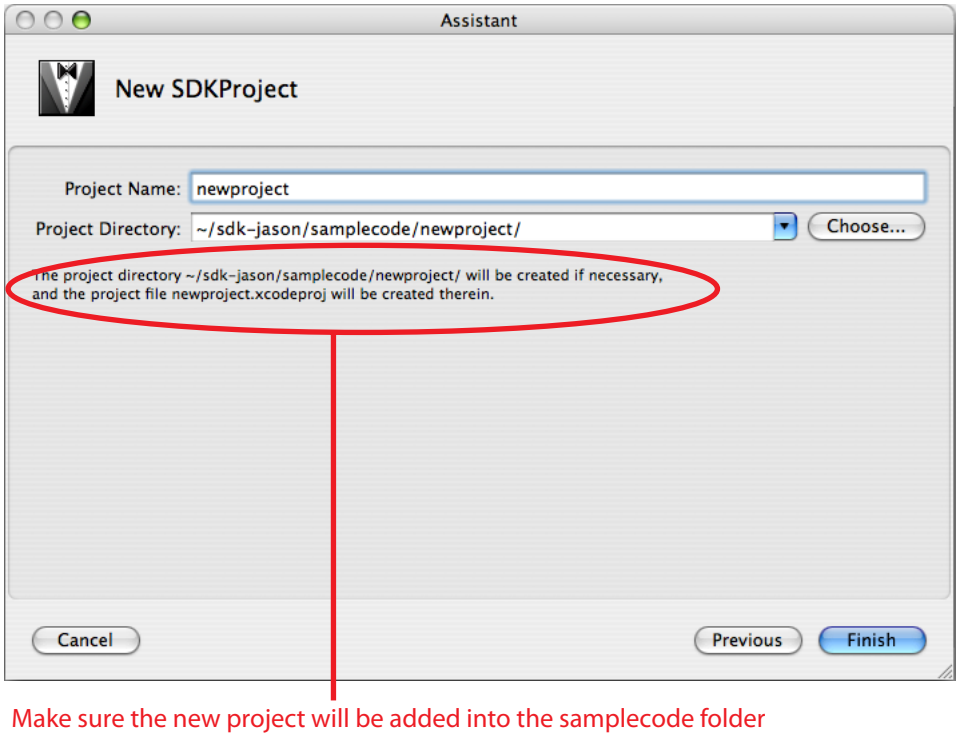


FIGURE 7 ***Illustrator Project Template Assistant for Xcode***



Notes:

- If you prefer to create a project outside the samplecode folder, remember to change the relative paths. See "Updating Relative Paths" on page 23.
- If your project folder already exists, point Xcode to its parent directory. Xcode looks for the folder before creating it.
- If a "Files to be Overwritten" dialog appears with an invisible .DS_store file shown as the existing file, click "Overwrite Selected Files."

Your new or existing project folder now contains the Xcode project file with the name you specified, and the new project is opened automatically.

All files selected to be overwritten are retained and renamed with a .moved-aside extension. If a .DS_store file is the only file overwritten, you can safely discard the .moved-aside file that appears in the project folder.

The new project has the same settings as an SDK sample project, except for those settings that are unavailable until source code is added. It contains common files needed for all SDK projects: Carbon.framework in the Libraries folder, Info.plist, about.rsrc, and PiPL.rsrc in the Resources folder, and a placeholder for the product in the Products folder.

Adding Existing Source Files to the Project

The next step is adding your own C++ and other source files, as well as your own and SDK-defined resource files, to the new project.

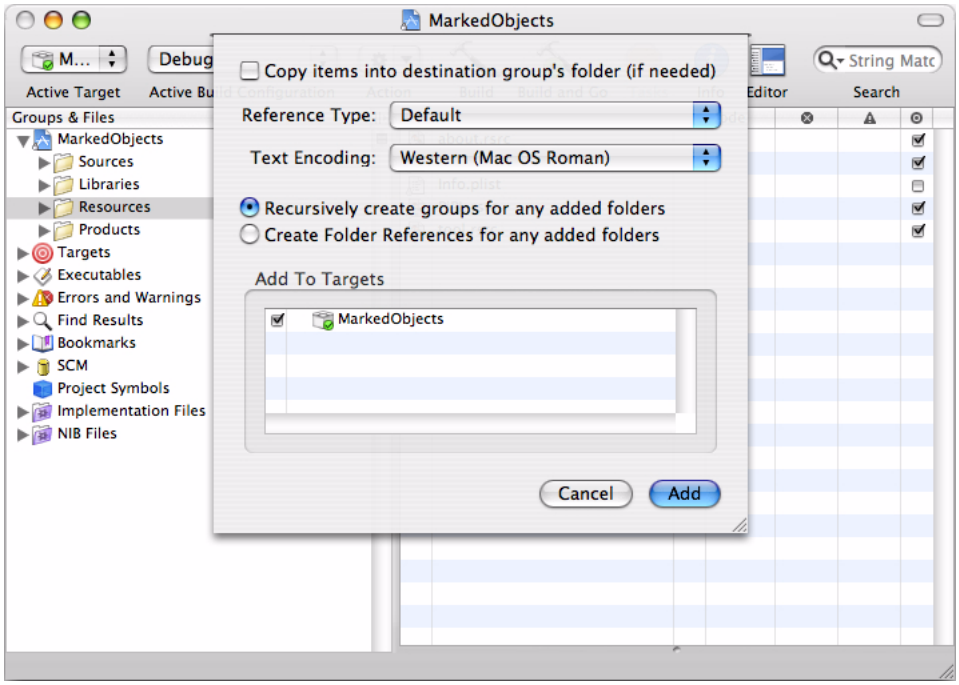
To add source files, you can either right-click the Sources folder and browse to the files, or drag and drop all of them into the Sources group under Groups & Files in the Xcode project window. Use the same techniques to add resource files to the project's Resources group.

- Add your plug-in's existing source files to the Xcode project's Sources group.
- Add the common source files, like shellMain.cpp, found in <*SDK*>/samplecode/common/source.
- Add your plug-in's existing resource files to the Xcode project's Resources group.
- Add the common resource files, like PiPL.rsrc, found in <*SDK*>/samplecode/common/mac.

Xcode prompts for whether files should be added to the target; the box is selected by default. See Figure 8.

- If you do not want the source files added to the target, deselect the box. Keep in mind, however, that only files added to the target are processed during the build.
- Add the resource files to the target if you want them to be bundled into the product.

FIGURE 8 ***Adding a Resource to the Target***



If your project needs to search header files in places other than the default SDK header search paths, add the paths to the project's Header Search Paths setting.

Expand the projects target in the Xcode project window to reveal the five build phases for the target. You should notice each source file you chose to add to the target is listed in the "Compile Sources" build phase, and each resource file you chose to add to the target is listed in the "Build ResourceManager Resources" build phase, along with about.rsrc and PiPL.rsrc.

Adding New Files to a Project

To add new files to your project, choose File > New File... and select the appropriate type of file. Give the file a name (preferably without spaces), and save it.

By default, the check box to add a new file to the target is checked. If you do not want the file to be processed when building the target, uncheck the box.

In the project window, store the new files in the appropriate group folders.

Updating Relative Paths

If you do not want to create your project in the <*SDK*>/samplecode/ directory, you can still use the template, but you must update the relative paths to Info.plist, about.rsrc, and PiPL.rsrc. To do this, follow these steps:

1. Control-click on the file, and select Get Info.
2. In the dialog, select Choose and navigate to the folder where the file is located.

You also must update the following project settings that contain relative paths:

- Header Search Paths — In particular, the paths to <*SDK*>/illustratorapi/ and <*SDK*>/samplecode/common/.
- Info.plist File.
- Info.plist Preprocessor Prefix File.
- Prefix Header — This must be set for both debug and release configurations.

NOTE: If there are spaces anywhere in your relative path, you must enclose the full path in double quotes. Xcode uses spaces as a path delimiter. See "Why do I get linker errors for projects whose path includes spaces?" on page 29.

Configuring Project Settings

After the source and resource files are added to your project, several additional settings are available in the Project Get Info dialog. To configure these setting, follow these steps:

1. With the project icon highlighted, select Info, then select the Build tab.
2. Make sure All Configurations is selected in the Configuration box, and All Settings is selected in the Collection box.

3. Change these settings:

- Set “Compile Sources As” to C++.
- Check “Precompile Prefix Header.”
- Check “Increase Sharing of Precompiled Headers.”
- Set “Level of Debug Symbols” to Referenced Symbols [used,-gused].
- Check “Symbols Hidden by Default.”

4. Enable these warnings:

- Missing Function Prototypes
- Unused Variables
- Unknown Pragma

5. Select Debug in the Configuration box, and set “Optimization Level” to None [-O0].
6. Set “Prefix Header” to ./../common/includes/IllustratorSDKdebug.pch.
7. Select Release in the Configuration box, and set “Optimization Level” to Fastest [-O3].
8. Set “Prefix Header” to ./../common/includes/IllustratorSDKrelease.pch.

Building an Illustrator SDK Xcode Project

After adding the source files and updated the project settings, you can build your project by pressing cmd-B or choosing Build > Build.

By default, Xcode stops the build at the first occurrence of an error. If you do not want this behavior, select the Xcode > Preferences... > Building tab. Under Build Options, check “Continue Building After Errors.”

The build status is displayed in the bottom of the project window, as shown in Figure 9. If there is a warning or error, the bottom right corner of the project window shows a tiny icon, yellow for warning or red for error. Click the icon or choose Build > Build Results to bring up the Build Results window, as shown in Figure 10.

In the Build Results window, use the small icons to show or hide different kinds of information. For instance, showing the transcript gives you more information about the error. By default, Xcode hides any existing warnings from a prior build. If these exist, click the small yellow triangle icon to show them.

FIGURE 9 ***Build Status in the Project Window***

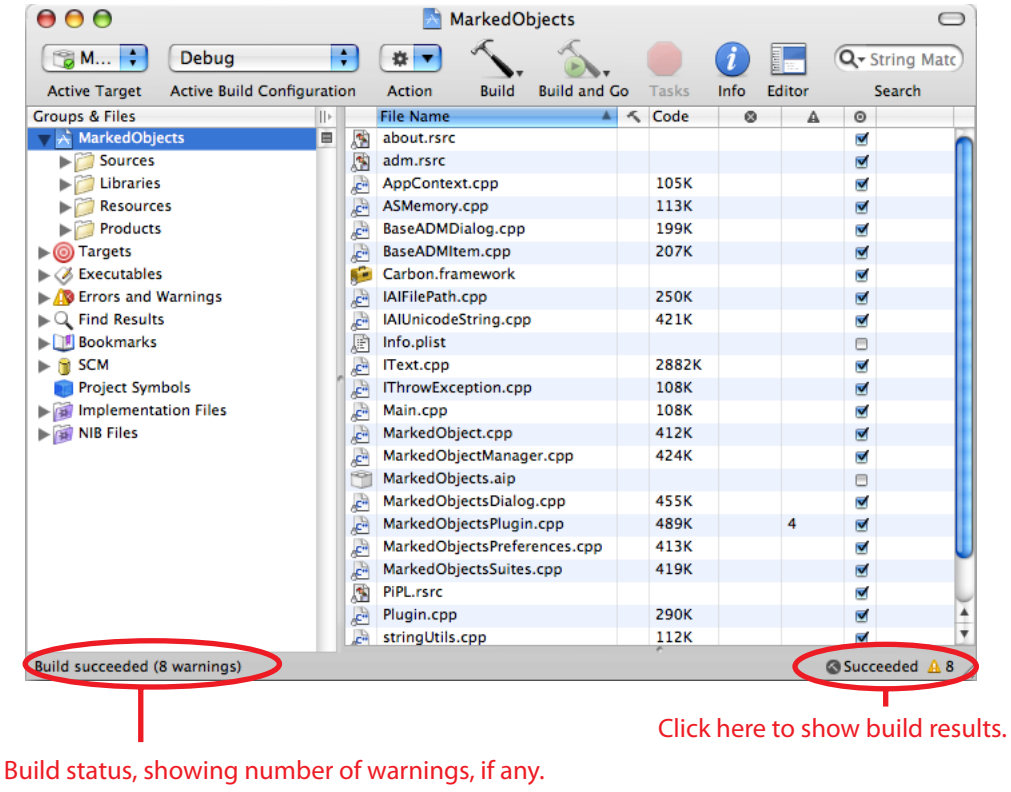
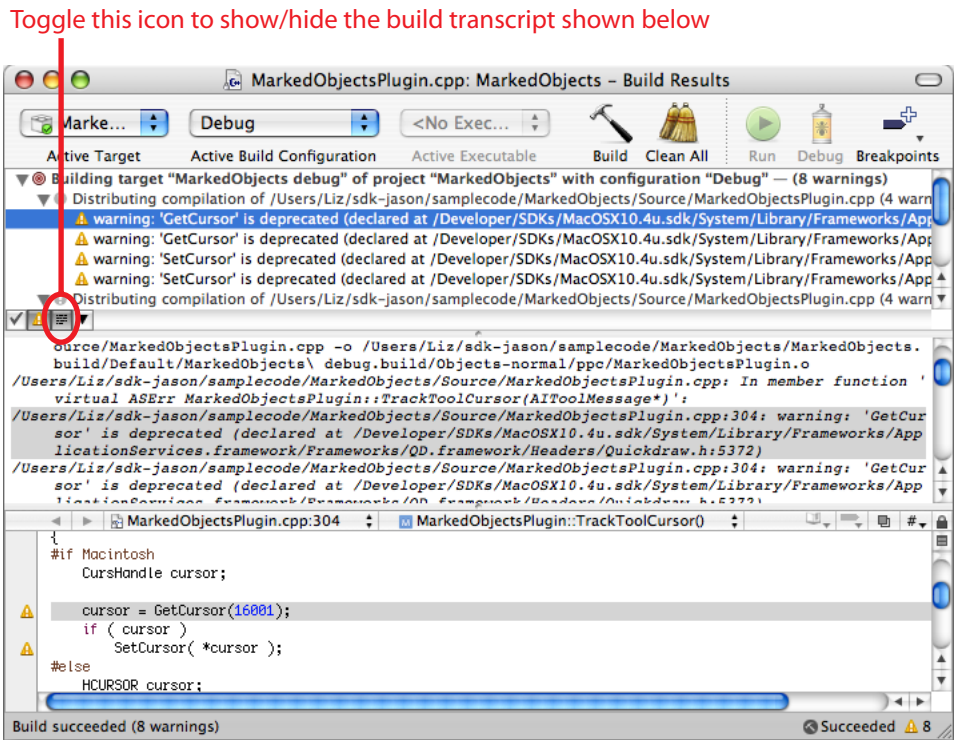


FIGURE 10 ***Build Result Window***



Known Xcode issues

error: expected constructor, destructor, or type conversion before '(' token

This occurs because of the Microsoft specific *__declspec(dllimport)* directive used by the plug-in entry-point declaration in sample code on the Illustrator CS2 SDK:

```
#ifdef MAC_ENV
      #define PluginMain main
#endif

#define DllExport extern "C" __declspec(dllexport)
DllExport SPAPI SPErr PluginMain( char *caller, char *selector, void *message );
```

To fix this remove the *__declspec(dllimport)* directive from your entry point declaration on Macintosh. See "PlugInMain Function Declaration Changed" on page 38 for the recommended approach.

error: Expected 'SYSHEAP', 'APPHEAP', 'PURGEABLE', 'NONPURGEABLE', 'LOCKED', 'UNLOCKED'

This can occur if you have files in the Build ResourceManager Resources build phase for which there is no rule. You can remove such a file by right-clicking the file and selecting Delete. When prompted, be sure you delete only the references. Alternately, you can add the necessary rule into the Rules tab.

This issue was encountered in the SDK when there was a .plc file in this build phase and no rule instructing the compiler on how to deal with this type of file. This file and rule are no longer required and were removed.

error: 'n' is not declared in this scope

The ISO C++ standard states: "an explicit specialization shall be declared in the namespace of which the template is a member, or, for member templates, in the namespace of which the enclosing class or enclosing class template is a member."

If you get this error, check for code like the following:

```
template <class T>
struct Something{
 struct B {
 int n;
 };
 struct D : B {
      void f () {
      n = 0; // ERROR
      }
 };
};
main(){
 Something<int>::D x;
 x.f ();
 return 0;
}
```

The problem is that *n* does not explicitly depend on *T*, so GCC looks up the name *n* at the point of definition. Using *this->n* instead solves the problem, because it informs the compiler that *n* depends on *T*; look-up is deferred to the point of instantiation, where the actual base class is visible.

error: exception handling disabled, use -fexceptions to enable

If your project uses exception handling, you can turn on the exception within the plug-in scope by turning on Enable C++ Exceptions in the project/target setting. For local-file scope exception handling, see "How can I turn on exceptions for just one file?" on page 29 in FAQ.

warning: comparison is always false due to limited range of data type

This warning is caused when you declare a variable as a particular type (for example, a short), then compare it to a variable or value of a different type (for example, an int). Change the type declarations so they match, an int to an int, a short to a short. Remember, it always is preferable to keep variable types as small as possible to save memory.

warning: throwing NULL, which has integral, not pointer type

This warning is caused by the following type of statement:

```
if (something == NULL) throw (NULL);
```

Change the throw to a return; for example:

```
if (something == NULL) return;
```

warning: converting to 'int32' from 'double'

Try to match types during assignment. For example, if an integer value is stored in a double variable, change the variable to be an integer if possible.

samplecode/YourProject/../../IllustratorAPI/Illustrator/IText.cpp: No such file or directory

API files for the Adobe Text Engine (ATE) have moved to a different folder; see "Adobe Text Engine API Files Moved" on page 36.

This is reported for any ATE file used in your project from the old ATE location. Frequently used files are IText.cpp and IThrowException.cpp.

warning: 'GetCursor'/'SetCursor' is deprecated

You may see warnings from the MAC OS X 10.4u SDK about deprecated methods, like GetCursor and SetCursor. Replacing these is optional; they are not yet replaced in the SDK.

Frequently Asked Questions

Why does my resource not seem to get built correctly?

All .r files can be compiled by Rez in the Build ResourceManager Resources phase. If you have a different type of resource file, you must add the file in the Compile Sources phase and add a build rule for that type of file. If you have a precompiled .rsrc file, it should be added in the Build ResourceManager Resources phase.

Why does my project fail to link against a library?

If your target is failing to link against a library that you put in the project, look under the Targets node (bullseye). Under the target's name are the five build phases. Expand "Link Binary With Libraries," and see if the library you want to link against is there. If not, drag it there.

What is the SDKROOT setting in Xcode?

It is important to distinguish SDKROOT from the Illustrator SDK root folder. The SDKROOT setting in the SDK Path of the project or target settings specifies the path to the Mac OS system SDK, not the Illustrator SDK.

Illustrator is cross-developed using the current Mac OS SDK, which is specified in the project setting's General tab. The Illustrator deployment target (MACOSX_DEPLOYMENT_TARGET) is set to Mac OS X 10.4.

Why are some of the variable values in the Debugger window set to "Incomplete Type"?

If the variable is a built-in data type, try setting the "Level Of Debug Symbols" setting (GCC_DEBUGGING_SYMBOLS) to "All Symbols [full, -gfull]." Illustrator SDK samples set the setting to "Referenced Symbols [used, -gused]." If the variable is a custom data type, you might have to write a custom data formatter. See "Formatting Program Data for the Debugger" on page 34.

How can I turn on exceptions for just one file?

If a CPP file uses #pragma exceptions (or includes a header file that does this), right-click on the CPP file, select Get Info, then type this in the Build tab:

```
-fexceptions
```

This adds an additional compiler flag that turns on exceptions for just that file.

Is #pragma message supported in Xcode?

No. If you are using #pragma message in CodeWarrior, try using #warning instead; for example:

```
#warning "This is a warning"
```

Why do I get linker errors for projects whose path includes spaces?

If you are adding a relative path that contains spaces, you must surround the path with double quotation marks; otherwise, you get linker errors because the path is broken into separate chunks that typically point to nowhere. In fact, if you click Library Search Paths or Framework Search Paths, the Xcode note at the bottom of the Info window indicates this: "Paths are delimited by white space, so any paths with spaces in them need to be properly quoted."

How do I resolve undefined symbols link errors?

The default Illustrator SDK sample settings do not tell you what code is referencing the missing symbols. By default, when you get an ld:Undefined Symbols error, you cannot tell where the error message comes from.

In the Other Linker Flags setting, type -y followed by the symbol name and try to link again. Open the build transcript (click the tiny text icon on the error window), and the linker will tell you what is referring to the missing symbol.

What deployment target should be used in Illustrator SDK projects?

Illustrator CS3 supports Mac OS 10.4 and later. You should either set your Mac OS X Deployment Target ($MACOSX_DEPLOYMENT_TARGET) to 10.4 in the project-level settings or, if you prefer code to load on any Mac OS system that supports the used functions, set it to the compiler default.

Use the per-architecture variants of the MACOSX_DEPLOYMENT_TARGET to specify 10.4 as the deployment target. For example, set MACOSX_DEPLOYMENT_TARGET_i386 and MACOSX_DEPLOYMENT_TARGET_ppc to 10.4.

Set the SDK path to /Developer/SDKs/MacOSX10.4u.sdk.

Is there an equivalent to the build-system macro definition, such as __MWERKS__?

In CodeWarrior there is a #define __MWERKS__ that is used in some SDK samples to determine which IDE is being used to build the plug-in. For Xcode-specific code, you can use this:

```
#ifdef __GNUC
```

For more information, see the following document: http://developer.apple.com/documentation/DeveloperTools/Conceptual/MovingProjectsToXcode/migration_differences/chapter_2_section_25.html#//apple_ref/doc/uid/20001709/BABFEJJG.

How do I maintain parallel projects in Xcode and CodeWarrior?

See the "Maintaining parallel projects in Xcode and CodeWarrior" chapter in the Apple document *Porting CodeWarrior Projects to Xcode*, available at the following location: http://developer.apple.com/documentation/DeveloperTools/Conceptual/MovingProjectsToXcode/index.html.

Is there an easy way to change a project type?

It is unlikely you will need to change the project type, if you follow the guidelines in this guide to convert a CodeWarrior project or create a new Xcode project for building Illustrator plug-ins. If you want to do so, open the pbxproj file in the Xcode project package. There is a productType value, listed like this:

```
productType = "com.apple.product-type.bundle"
```

You can, for example, change this to "com.apple.product-type.framework" and re-open the project. This changes the project from the bundle to the framework project type. You can experiment with other changes.

What should I do if I get this debugger error: "Previous frame inner to this frame (corrupt stack?)"

If you see this error when stepping through code in the Xcode debugger, see "Problems Stepping Through Code in the Xcode Debugger" on page 34 for workarounds.

Tips for Working with Xcode

- There are some preferences that are not in the Xcode interface. For information on additional options you can set, see http://developer.apple.com/releasenotes/DeveloperTools/Xcode/XcodeDefaults.html. For example, you can set how many recent projects you want Xcode to remember.
- If you have a source file open in Xcode, you can Option+double-click a system call to look up its documentation, or Command+double-click any interface item to open the file in which it is defined (e.g., Cmd+double-click IApplication to open IApplication.h).
- When you open a file in Xcode, there is a little splitter widget above the scrollbar on the right side. Click the splitter to split the window vertically. Command+click the splitter to split the window horizontally.

- By default, each time you open a .h or .cpp file, a new editor window opens. This is the Grouped mode. In the toolbar of the editor window, click Ungrouped to toggle Grouped/Ungrouped mode. In Ungrouped mode, if you open a file, it opens in the same editor window. Navigate between open files by clicking the drop-down list that contains the filename and current line number.
- Xcode has a layout preference (Xcode > Preferences > General) with three options: Condensed (CodeWarrior-like), Default (Xcode's default), and All-In-One (Visual Studio-like).
- Xcode has configurable keyboard shortcuts (Xcode > Preferences > Key Bindings) and ships with several sets: Xcode defaults, MPW-like, CodeWarrior-like, and BBEdit-like.
- If you do not like the default Xcode window size, you can configure the windows in one project and save that as the default. For instruction on how to save the layout, search Xcode documentation for "Save Changes to the Current Layout."

References

The following documents are available at http://developer.apple.com/documentation/DeveloperTools/Xcode-date.html:

- *Xcode Quick Tour Guide*
- *Porting CodeWarrior Projects to Xcode*
- *Xcode 2.4 User Guide*
- *Framework Programming Guide*
- *Xcode Build Setting Reference*

Running and Debugging

Getting Illustrator to Load your Plug-in

We recommend you use the Additional Plug-ins Folder preference when developing and debugging plug-ins. This preference is set in the application's Preferences > Plug-ins & Scratch Disks... dialog. For a complete list of the locations where Illustrator looks for plug-ins, see *Adobe Illustrator CS3 Programmer's Guide*.

Debugging a Plug-in under Visual Studio

To debug your plug-in, follow these steps:

1. Make sure your plug-in is in a location where Illustrator can find it (see "Getting Illustrator to Load your Plug-in" on page 31).
2. Open your plug-in project in Visual Studio.

3. Set a breakpoint in your code. If you are getting started, set the breakpoint in your plug-in's entry-point function, PluginMain.
4. Start the debug session using Visual Studio's Debug > Start Debugging menu.
5. An Executable for Debug Session dialog appears. Visual Studio must be able to find the Illustrator application. In the Executable filename field, enter or browse to the path to the Illustrator executable. The path to the default install location is as follows:

```
C:\Program Files\Adobe\Adobe Illustrator CS3\Support Files\Contents\Windows\
Illustrator.exe
```

6. Click OK, to start debugging your plug-in under Illustrator.

Debugging a Plug-in under Xcode

To debug your plug-in, follow these steps:

1. Make sure your plug-in is in a location where Illustrator can find it (see "Getting Illustrator to Load your Plug-in" on page 31).
2. Open your plug-in project in Xcode.
3. Set a breakpoint in your code. If you are getting started, set the breakpoint in your plug-in's entry-point function, PluginMain.
4. Add the Illustrator application as an executable to your plug-in's Xcode project, so Xcode can find the Illustrator application (see "Adding Illustrator to the Executable Group" on page 32).
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. This is similar to the Host Application in CodeWarrior.

To define the executable for your project, follow these steps:

1. Find the Executable group under the Groups & Files pane of the project window. Right-click Executables and choose Add > New Custom Executable from the context menu, as shown in Figure 11.
2. An executable set-up assistant appears, as shown in Figure 12. Give the executable a name, then specify the path to the application in Executable Path. The path to the default install location is:

```
/Applications/Adobe Illustrator CS3/Adobe Illustrator CS3.app
```

3. Select the project to add for the executable; this is the current project you are trying to debug.
4. Click Finish.

FIGURE 11 ***Adding a New Custom Executable for Debugging Purposes***

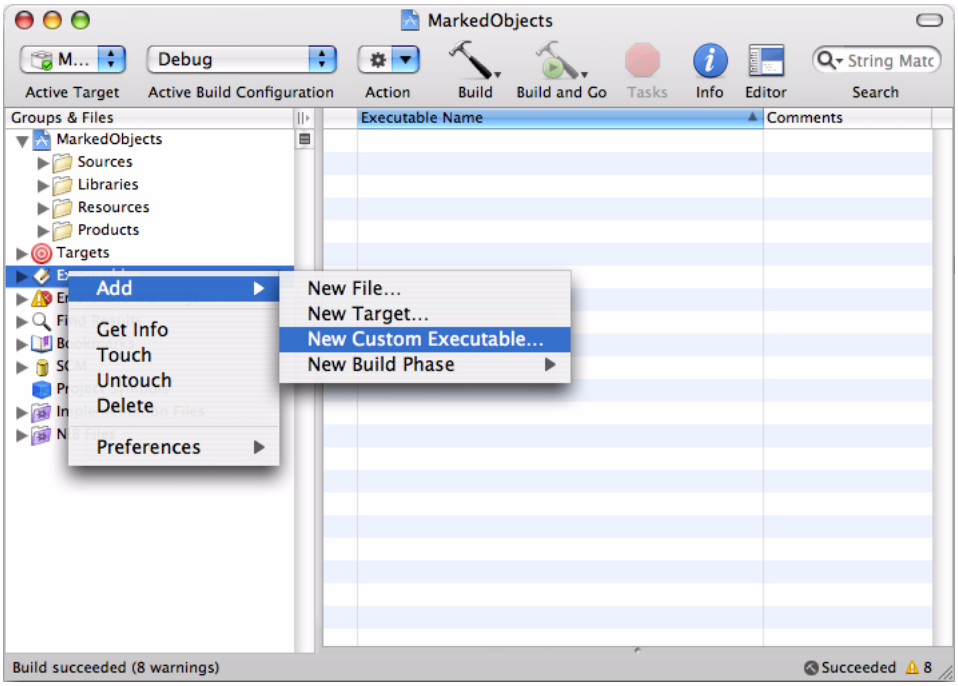
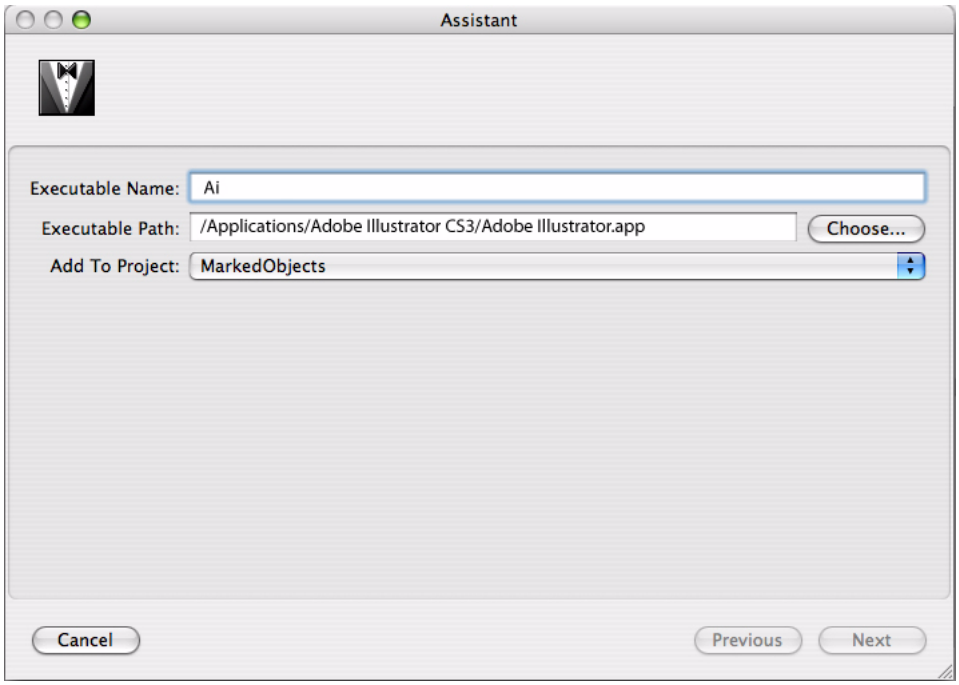


FIGURE 12 ***New-executable Assistant***



An Executable Info window appears that summarizes the executable environment you just created.

If you have only set up 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 "Getting Illustrator to Load your Plug-in" on page 31.

NOTE: If Illustrator is already running, you can use Build > Attach in Xcode to attach the debugger and current project to the running process, if you 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 the 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 to decide on 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 Debug > 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 make this easier, you can create a data formatter to customize how variables are displayed or summarized in the debugger. For more information, see http://developer.apple.com/documentation/DeveloperTools/Conceptual/XcodeUserGuide/Contents/Resources/en.lproj/index.html, and view the chapter on Examining Program Data and Information.

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 first make sure you are using the supported version of Xcode (see "Development Platforms" 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 the following 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.

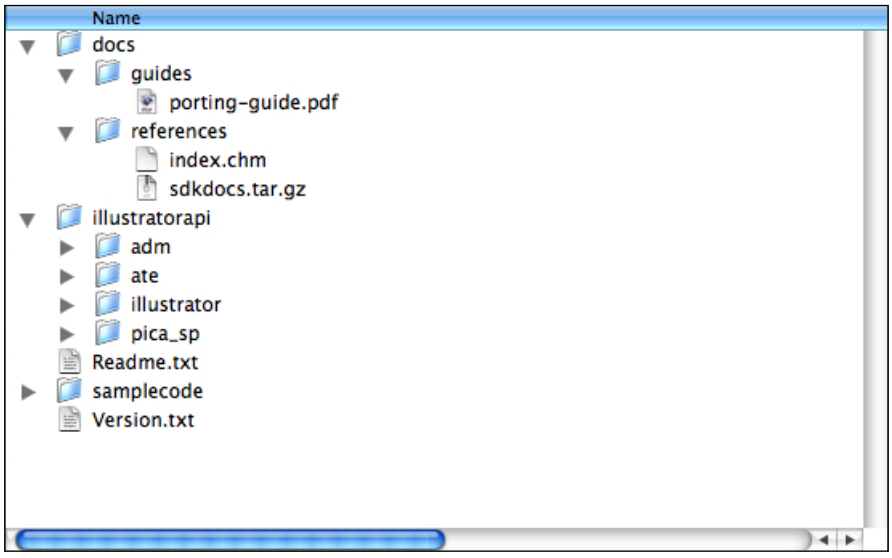
SDK Changes

Organizational Changes

The general folder structure of the Illustrator CS3 SDK has changed from the structure used for the Illustrator CS2 SDK; see Figure 13. These changes are described in more detail in the following sections:

- "Folder Name Case Changes" on page 36.
- "Documentation Changes" on page 36.
- "Adobe Text Engine API Files Moved" on page 36.
- "Change to the Structure of an SDK Sample Folder" on page 36.
- "Change to the Structure of the Output Folder" on page 37.

FIGURE 13 New SDK Folder Structure



Folder Name Case Changes

The names of the <*SDK*>/illustratorapi folder and the <*SDK*>/samplecode/common folder were changed to lowercase. The subfolders these folders contain also were similarly changed.

No project changes are required because of this change, since paths in Xcode and Visual C++ are case insensitive.

You can update your projects for this change if you wish. All SDK projects were updated to use the correct case.

Documentation Changes

The former Documentation folder is now called docs and contains two subfolders, one for guides and one for references. For instructions on using the reference documentation for the API, see "API Reference" on page 40.

Adobe Text Engine API Files Moved

The Adobe Text Engine (ATE) API files moved from the <*SDK*>/illustratorapi/illustrator/ folder to <*SDK*>/illustratorapi/ate/. The following files are affected:

```
ATEException.h
ATESuites.h
ATETextSuitesDeclare.h
ATETextSuitesExtern.h
ATETextSuitesImplement.h
ATETextSuitesImportHelper.h
ATETypes.h
ATETypesDef.h
IText.cpp
IText.h
IThrowException.cpp
IThrowException.h
```

You need to make the following changes to your projects:

- *Visual Studio projects must be updated to add the new folder as a path to search for include files.* See "fatal error C1083: Cannot open include file: "ATETypesDef.h": No such file or directory" on page 10.
- *If your projects use any of these files, you must add them from their new locations.* For Visual Studio see "fatal error C1083: Cannot open source file: '..\..\illustratorapi\illustrator\IText.cpp': No such file or directory" on page 10. For Xcode see "samplecode/YourProject/../../IllustratorAPI/Illustrator/IText.cpp: No such file or directory" on page 28

The SDK samples affected by this change are MarkedObjects, TextRoadMap, TextTest, and TextFileFormat.

Change to the Structure of an SDK Sample Folder

The Win, Mac and Common folders were removed from the project folder of each sample plug-in. They were replaced with the following structure:

- The project files, for Xcode and Visual Studio, are at the top level, <*SDK*>/samplecode/<*projectname*>/.
- The resource files are in the Resources folder, <*SDK*>/samplecode/<*projectname*>/Resources/.
- The source files are in the Source folder, <*SDK*>/samplecode/<*projectname*>/Source/.

Change to the Structure of the Output Folder

There is a slight change to how the output folder is structured when a plug-in is built:

- When a project is built in Visual Studio, the product is created in <*SDK*>/samplecode/output/win/debug.
- When a project is built in Xcode, the product is created in <*SDK*>/samplecode/output/mac/debug or <*SDK*>/samplecode/output/mac/release, depending on the configuration specified.

File shell(common).cpp Renamed as shellMain.cpp

The file was renamed because Xcode/gdb does not honor breakpoints set in functions within shell(common).cpp. The parenthesis characters in the filename are the cause of the problem. They are interpreted as something other than filename characters in gdb's interface.

The shell(common).cpp file contains the PluginMain function (the plug-in entry point) for SDK plug-ins that use the Shell framework. Several sample plug-ins are based on this framework (ADMNonModalDialog, MultiArrowTool, Shell, etc.). If your plug-in used shell(common).cpp, remove that file from your projects and add shellMain.cpp instead.

File Pcmain.cpp Removed from the SDK

The DllMain function defined by this file is an optional entry point into a dynamic-link library (DLL) on Windows that can contain initialization and termination code. For more information, see Microsoft's MSDN documentation. The SDK samples do not need code of this nature, so the file was removed from the SDK.

PIDebugCarbon.h/PICarbon.h Removed from the SDK

The prefix header files PIDebugCarbon and PICarbon used in earlier SDKs to compile code on the Macintosh under CodeWarrior were removed. The prefix header file used for compilation under Xcode is PluginStd.h. It is included by the files used to build the pre-compiled headers for Xcode. It is unlikely your projects were using the deleted files directly. If you were, see the following files for the set-up on the current SDK:

- <*SDK*>/samplecode/common/includes/IllustratorSDKDebug.pch
- <*SDK*>/samplecode/common/includes/IllustratorSDKRelease.pch
- <*SDK*>/samplecode/common/includes/PluginStd.h

File shell(common).rsrc Removed from the SDK

This file contained a legacy version ("vers") resource that is no longer required by Illustrator SDK plug-ins for the Macintosh. Version information displayed by the Finder's Get Info window is now specified by an Info.plist file. See "Plug-in Version Information Added to the SDK" on page 39.

PlugInMain Function Declaration Changed

The PluginMain entry-point function declaration was changed for the following reasons:

- Xcode will not compile the old declaration; the Windows-specific __declspec(dllexport) compiler directive causes an error to be thrown. See "Known Xcode issues" on page 26.
- The entry-point function name is different on Mac OS (main) and Windows (PluginMain). With the new approach below, a single function named PluginMain is used on both platforms. See Example 1 and Example 2.

EXAMPLE 1 Old PluginMain Declaration

```
#ifdef MAC_ENV
      #define PluginMain main
#endif

#define DllExport extern "C" __declspec(dllexport)
DllExport SPAPI SPErr PluginMain( char *caller, char *selector, void *message );
```

EXAMPLE 2 New PluginMain Declaration

```
// 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
```

This change affects the following files:

```
Plugin.hpp
Main.cpp
shellMain.cpp
Tutorial.cpp
```

If your plug-in uses any of the files above to declare the PlugInMain function, the change was made for you. If you created your own file that declares this function, you may have to make this change. You also may have to adjust your project settings as outlined below.

On Windows, a module definition (.def) file is now used to export the entry-point function from the DLL. This replaces the __declspec(dllexport) compiler directive in the old declaration. The Linker > Input > Module Definition File property in the Visual Studio project settings for SDK samples now refers to the <*SDK*>/samplecode/common/win/PluginMain.def file.

On Mac OS, the gcc compiler pragmas shown in Example 1 are used to make the plug-in entry point visible. The Xcode project settings for SDK samples have the Symbols Hidden by Default property turned on (GCC_SYMBOLS_PRIVATE_EXTERN = YES;). The pragmas override this setting for the PluginMain function; this allows Illustrator, or more accurately the dynamic linker, to find this function in the plug-in. For more information, see the URLs below:

`http://developer.apple.com/technotes/tn2007/tn2185.html`
`http://developer.apple.com/documentation/DeveloperTools/Conceptual/CppRuntimeEnv/Articles/SymbolVisibility.html`

PiPLs Provided in Resource Source Code

In the Illustrator CS2 SDK, PiPL resource files for sample plug-ins were provided in a binary form. These binary files continue to be provided:

`<SDK>/samplecode/common/mac/PiPL.rsrc`
`<SDK>\samplecode\common\win\PiPL.bin`

The sample plug-ins, however, no longer use these files. The sample plug-ins now define their PiPLs in a resource source-code form. For example, the Tutorial plug-in's PiPL is defined in the following source files:

- Windows — See the PiPL resource declaration in the Tutorial.rc file.
- Mac OS — See the PiPL resource declaration in the Tutorial.r file.

It is more flexible to work with PiPL resources in source-code form. You can specify the name of your plug-in within that resource, for example. The binary PiPL files provided in the SDK leave the plug-in name property blank, so they can be used by any plug-in. For further background documentation on PiPLs, see *Adobe Illustrator CS3 Programmer's Guide*.

Plug-in Version Information Added to the SDK

The .aip plug-in produced by each SDK project now contains version information, viewed by right-clicking on the file and selecting Get Info (Macintosh) or Properties (Windows). Included in this information is copyright, company, file name and version details.

The source for this information is <*SDK*>/samplecode/common/includes/SDKDef.h.

In Xcode, the version information is provided through the Info.plist file included in all SDK plug-ins and the template. This file is located in <*SDK*>/samplecode/common/mac.

In Visual Studio the version information is added to the projects .rc file by including VersionInfo.rc, located in <*SDK*>/samplecode/common/win

New pragma.h file Added to the SDK

This new file was created in response to a warning in Visual Studio, was documented in "Known Visual C++ 2005 Issues" on page 9. See that section for details of when and how to use this file.

Precompiled Headers Set-up Changed on Macintosh

Due to the change to Xcode, the set-up of the precompiled headers has changed. When an SDK sample is built, a new folder called SharedPrecompiledHeaders is created in <*SDK*>/samplecode/output/mac/. Inside this folder are the precompiled headers, one for the debug configuration and one for release. Sample plug-ins are set up to share these precompiled headers.

Illustrator API Changes

API Reference

Reference documentation for the API can be found at these locations:

- <*SDK*>/docs/references/index.chm — This is the API reference in compiled HTML format. This searchable help file contains reference documentation for the API. To view the contents on Windows, double-click the index.chm file icon in the Windows Explorer to open the home page. To view the contents on Mac OS, you need a CMH viewer (see Table 2 for some options).
- <*SDK*>/docs/references/sdkdocs.tar.gz — This is an archive of API reference HTML files. This archive contains HTML reference documentation for the API. After installing the SDK, double-click the sdkdocs.tar.gz file to decompress the archive. View the documentation in your browser.

API Advisor

A summary of the changes made to the low-level API between CS2 and CS3 is provided in the <*SDK*>/docs/references/apiadvisor-ai12-vs-ai13.html file. The report describes classes, structures, and files that were added, removed, or altered. Details are listed for function and function pointer changes.

PiPL Changes on Macintosh

Illustrator requires that plug-ins built to run on an Intel Macintosh platform have a new PiPL property which contains the code descriptor for that platform.

PiPLs

A PiPL (Plug-in Property List) is a resource file that contains static information about a plug-in. It is required by all Illustrator plug-ins. The main role of the PiPL is to tell the plug-in loader where the code is—the plug-in entry point. Illustrator cannot load a plug-in that does not have a PiPL resource; if it cannot find the resource, it ignores the plug-in. For further background documentation on PiPLs, see *Adobe Illustrator CS3 Programmer's Guide*.

PiPL Project for Xcode

Previously, you could view or edit the information in the PiPL only by opening the .rsrc file in a resource editor like Resorcerer.

The SDK now provides a new Xcode project, <*SDK*>/samplecode/pipl/pipl.xcodeproj, that uses Rez to compile two source files, Plugin.r and PiPL.r, to create the PiPL.rsrc file. This PiPL.rsrc file is built into the <*SDK*>/samplecode/common/mac folder, where it is included by all SDK projects.

The <*SDK*>/samplecode/pipl/PiPL.r file defines all available properties. The PiPL used in the SDK plug-ins does not use all of these. The <*SDK*>/samplecode/pipl/Plugin.r file lists the properties it uses and builds into PiPL.rsrc.

Code-descriptor Property Changes

A code-descriptor property informs Illustrator of the plug-in type and the name of the plug-in entry function. In CS2, the PiPL.rsrc file contained the code-descriptor properties ppcb (for a PEF plug-in on a PowerPC machine) and mach (for a Mach-O plug-in on a PowerPC machine). In CS3, with Macintosh's latest move to Intel hardware, a new code-descriptor property, mi32, was added. The mi32 property describes the x86 Intel code for the plug-in.

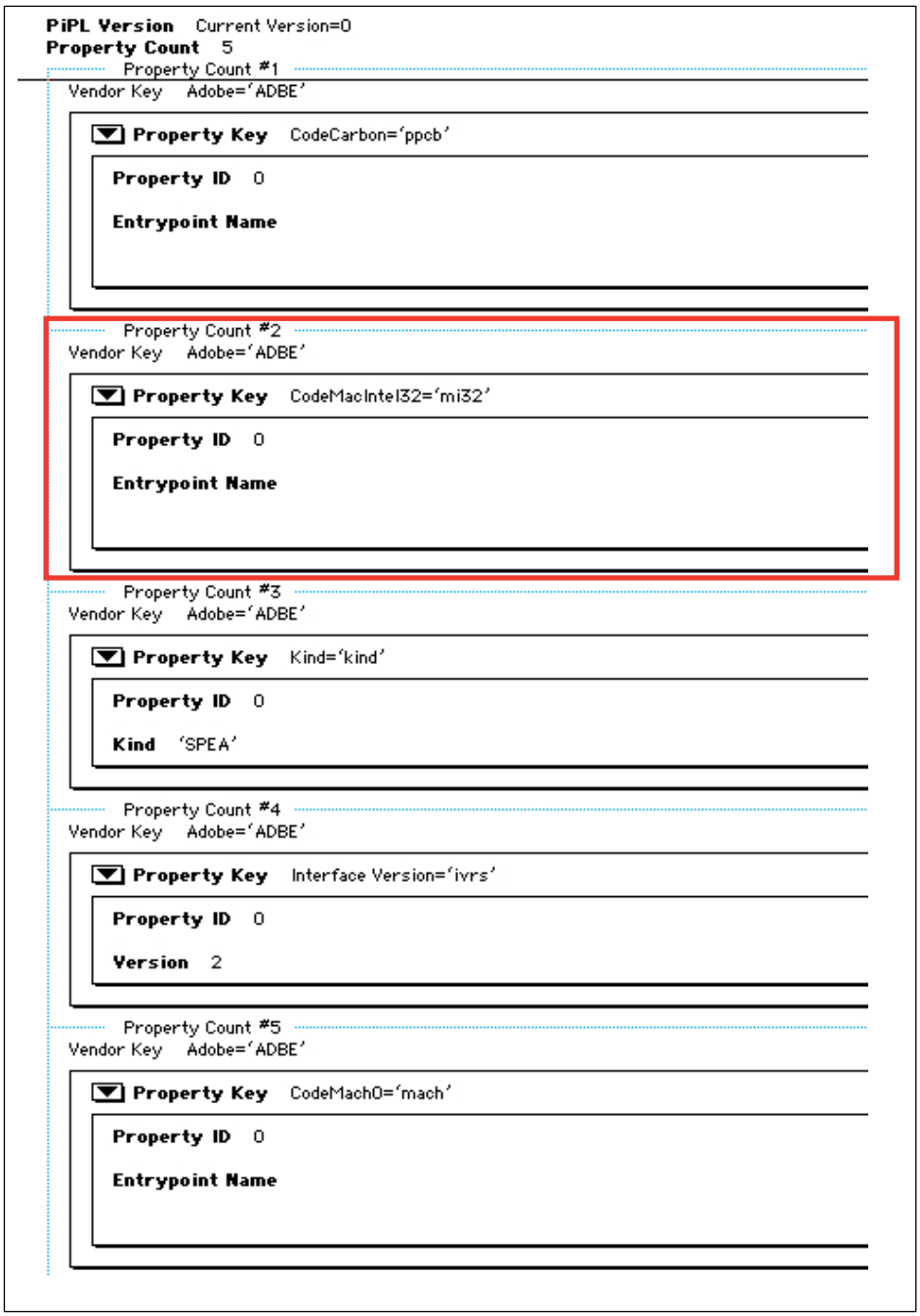
The PiPL.rsrc file in <*SDK*>/samplecode/common/mac already contains the new code descriptor. If your project used the PiPL.rsrc file provided by a previous version of the SDK, we recommend you update your project to use the new version.

If your project is set up to build your PiPL resource from source, use PiPL.r and Plugin.r in <*SDK*>/samplecode/pipl as guidelines for updating your equivalent file(s).

A universal binary plug-in for Illustrator should have both an mi32 code-descriptor property *and* a mach property. On Intel Macintosh hardware, if your plug-in has a mach property *and does not have* an mi32 property, Illustrator will load your plug-in only when running via Apple's Rosetta translator. Rosetta enables code compiled for PowerPC processors to run on Macintosh systems that use Intel processors.

Figure 14 shows the new PiPL.rsrc file, viewed with Resorcerer.

FIGURE 14 ***PiPL.rsrc Viewed in Resorcerer***



Byte-order Issues and PiPLs

Different operating systems use different byte-order formats:

- On PowerPC machines, the memory format has always been big-endian; that is, the most-significant bit (MSB) is stored at the lowest address in memory and is accessed first.
- On Windows and Intel Macintosh machines, the memory format is little-endian: the MSB is stored at the highest address in memory, and is accessed last.

On these different machines, the same data is read and stored differently, with information saved to disk on one being seen as backwards on the other. This does not cause a problem in Windows, because the PiPL is built in Windows. Macintosh plug-ins, however, are built as universal, meaning they can be loaded on either Intel or PowerPC Macintosh hardware.

Illustrator expects the data in the mi32 code-descriptor property to be in big-endian byte order; it swaps the bytes when reading them for Intel Macintosh. If you follow the guidelines given here for creating your PiPL from .r source code using Rez, this works correctly. In Xcode, Rez writes all multi-bye data in the big-endian format; however, If you use a tool like Resorcerer to edit your existing PiPL binary, you must be sure to use big-endian byte order.

Applying a Paragraph or Character Style to Text in the Result Group of a Plug-in Art Item

A bug in Illustrator CS1 and CS2 caused text objects created inside the result group of a plug-in art object using the AITextFrameSuite functions below to malfunction if the user invoked undo/redo commands across the creation of that text object:

```
AITextFrameSuite::NewPointText()
AITextFrameSuite::NewInPathText()
AITextFrameSuite::NewOnPathText()
AITextFrameSuite::NewOnPathText2()
```

This bug is fixed in Illustrator CS3; however, the fix exposes a limitation with the text API that the previous releases avoided. The limitation is that the ITextRange functions below fails to apply a style to a range of text in text objects that are part of a result group:

```
ITextRange::SetNamedParaStyle()
ITextRange::SetNamedCharStyle()
```

For example, see Example 3.

EXAMPLE 3 Old Code

```
// Create a plug-in art result group
AIArtHandle art;
sArt->NewArt(kPluginArt, kPlaceAboveAll, NULL, &art);
AIArtHandle resultArt;
sPluginGroup->GetPluginArtResultArt(art, &resultArt);
```

```
// Create text objects in a result group
AIRealPoint point;
memset(&point, 0, sizeof(point));
AIArtHandle textFrame;
sAITextFrame->NewPointText(kPlaceInsideOnTop, resultArt,
kHorizontalTextOrientation, point, &textFrame);
TextFrameRef ateTextFrame;
sAITextFrame->GetATETextFrame(textFrame, &ateTextFrame);
ITextFrame iTextFrame(ateTextFrame);
ITextRange crange = iTextFrame.GetTextRange();
crange.InsertAfter("hello world");

// Apply an existing paragraph style
bool isApplied = crange.SetNamedParaStyle((ASUnicode *)L"mystyle");
// isApplied is always false meaning the style was not applied
```

The workaround is to retrieve the features from the desired style and apply them to the text range. See Example 4 and Example 5.

EXAMPLE 4 New code to Replace an ITextRange:: SetNamedParaStyle Call

```
// Get the current documents text resources
ATE::DocumentTextResourcesRef ateTextResourcesRef = NULL;
sAIDocument->GetDocumentTextResources(&ateTextResourcesRef);
IDocumentTextResources docTextResources(ateTextResourcesRef);
// To apply a paragraph style add:
// Retrieve features from the specified paragraph style
IParaStyle paraStyle = docTextResources.GetParaStyle((ASUnicode *)L"mystyle");
IParaFeatures paraFeatures = paraStyle.GetFeatures();
// Apply features to text range
crange.SetLocalParaFeatures(paraFeatures);
```

EXAMPLE 5 New Code to Replace an ITextRange::SetNamedCharStyle Call

```
// Get the current documents text resources
ATE::DocumentTextResourcesRef ateTextResourcesRef = NULL;
sAIDocument->GetDocumentTextResources(&ateTextResourcesRef);
IDocumentTextResources docTextResources(ateTextResourcesRef);
// Retrieve features from the specified character style
ICharStyle charStyle = docTextResources.GetCharStyle((ASUnicode *)L"mystyle");
ICharFeatures charFeatures = charStyle.GetFeatures();
// Apply features to text range
crange.SetLocalCharFeatures(charFeatures);
```

Getting a Plug-in Listed in the System Info Dialog

If your plug-in does not show up under the System Info dialog, the documentation below explains how to fix your code.

To be loaded by Illustrator, a plug-in must do the following:

- Use the correct Illustrator plug-in file extension (.aip).
- Have a well formed PIPL resource.

To show up in the System Info dialog, a plug-in must be loaded and do one of the following:

- Provide the plug-in name in its PIPL resource in the *pinm* property (the recommended approach - see *Adobe Illustrator CS3 Programmer's Guide*).
- Call SPPluginsSuite::SetPluginName on start-up.

About Plug-ins Dialog Removed

Before Illustrator CS3, the list of plug-ins that were loaded was presented in the About Plug-ins and System Info dialogs. The About Plug-ins dialog also allowed the user to view plug-in specific information. As of Illustrator CS3, the About Plug-ins menu and dialog are removed; the list of loaded plug-ins is presented in the System Info dialog.

The removal of the About Plug-ins dialog means Illustrator does not send the kSPInterface-AboutSelector selector to plug-ins anymore. If you want to display plug-in-specific information like company contact information or copyright statements, use an about plug-in menu as follows:

1. Create a new menu group under Illustrator's about group (see kAboutMenuGroup) which contains all your about plug-in menu items.
2. For each plug-in, create an about plug-in menu item under this new group.
3. Handle the menu message related to use of your about plug-in menu by popping an about box containing plug-in-specific information.

A helper class that supports this functionality is provided as sample code; see SDKAboutPluginsHelper. The Tutorial sample shows how this class is used.

Removal of bool Types from Suite APIs

Use of the bool type was removed from all suite APIs, to preserve binary compatibility. The bool type cannot be passed across a suite API into or out of Illustrator, because the size of a bool type is not standard; it varies from compiler to compiler. For example, in CodeWarrior a bool is 1 byte, and in Xcode a bool is 4 bytes.

Affected bool types were changed to a 1-byte type, AIBool8 or ATEBool8. You need to align the type of any affected booleans you cache in local variables accordingly.

Note that the C++ helper class APIs (such as IAIFilePath.cpp) still use bool types. This is intentional, as the helpers do not inhibit binary compatibility. In these files, the bool type is compiled on the client side and never passed across a suite API to Illustrator. A warning can be generated when compiling helper class APIs - see "warning C4800: "AIBool8" : forcing value to bool "true" or "false" (performance warning)" on page 9.

Porting Case Study

This section describes the steps required to port the MarkedObjects sample from the Adobe Illustrator CS2 SDK to the Adobe Illustrator CS3 SDK. Use this as an outline of the steps needed to port your own plug-in project.

Porting Marked Objects on Windows

This task was performed on a clean copy of the CS2 SDK for Windows:

1. Navigate to the MarkedObjectsProject/Win folder in the CS2 SDK.
2. Delete the .sln file. See "Initial Conversion" on page 7.
3. Start Visual Studio 2005.
4. Choose File > Open > Project/Solution.
5. Browse to MarkedObjects.vcproj, and open it.
6. When the Conversion window appears, select Next.
7. Choose whether to make a back up, then select Next. See "Initial Conversion" on page 7.
8. Review the details, and select Finish.
9. Choose to view the conversion log, and select Close.
10. After noting any warnings displayed in the conversion log, close the log. See "Conversion Log" on page 7.
11. Change the necessary project settings. See "Configuring the Project Settings" on page 8.
12. Make any necessary changes to the Additional Include Path. See "Adding Extra include Paths" on page 8.
13. Close the properties dialog.
14. Build the project. See "Building the Project" on page 8.
15. Fix compiler errors and warnings. See "Dealing with Visual Studio Warnings" on page 8, "SDK Changes" on page 35 and "Illustrator API Changes" on page 40 for guidance on frequently encountered issues.

Porting Marked Objects on Macintosh

This task was performed on a clean copy of the CS2 SDK for Macintosh, using the template recommended in the first paragraph of "Creating an Xcode Project for an Illustrator SDK Plug-in" on page 19:

1. Copy the template folder to the directory specified in "Setting up the Xcode Project Template for an Illustrator SDK Plug-in" on page 19.
2. Create the new project, as directed in Steps 1 - 4 of "Creating an Illustrator SDK Xcode Project from the Template" on page 19.
3. Add the source files to the Source folder and the resource files to the Resources folder in the Xcode project. See "Adding Existing Source Files to the Project" on page 22.
4. Update the relative paths as needed. (This step is required if you did not create your project in the <*SDK*>/samplecode/ folder.) See "Updating Relative Paths" on page 23.
5. Change the project settings. See "Configuring Project Settings" on page 23.
6. Build the project. See "Building an Illustrator SDK Xcode Project" on page 24.
7. Fix compiler errors and warnings. See "Known Xcode issues" on page 26, "SDK Changes" on page 35 and "Illustrator API Changes" on page 40 for guidance on frequently encountered issues.