

ADOBE® INDESIGN® CS5.5



ADOBE INDESIGN CS5.5 PORTING GUIDE



© 2011 Adobe Systems Incorporated. All rights reserved.

Adobe® InDesign® CS5.5 Porting Guide

If this guide is distributed with software that includes an end user agreement, this guide, as well as the software described in it, is furnished under license and may be used or copied only in accordance with the terms of such license. Except as permitted by any such license, no part of this guide may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, recording, or otherwise, without the prior written permission of Adobe Systems Incorporated. Please note that the content in this guide is protected under copyright law even if it is not distributed with software that includes an end user license agreement.

The content of this guide is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Adobe Systems Incorporated. Adobe Systems Incorporated assumes no responsibility or liability for any errors or inaccuracies that may appear in the informational content contained in this guide.

Please remember that existing artwork or images that you may want to include in your project may be protected under copyright law. The unauthorized incorporation of such material into your new work could be a violation of the rights of the copyright owner. Please be sure to obtain any permission required from the copyright owner.

Any references to company names in sample templates are for demonstration purposes only and are not intended to refer to any actual organization.

Adobe, the Adobe logo, Adobe Bridge, Creative Suite, InCopy, InDesign, Reader, and Version Cue are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries. Microsoft and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Macintosh and Mac OS are trademarks of Apple Computer, Incorporated, registered in the United States and other countries. All other trademarks are the property of their respective owners.

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA. Notice to U.S. Government End Users. The Software and Documentation are "Commercial Items," as that term is defined at 48 C.F.R. §2.101, consisting of "Commercial Computer Software" and "Commercial Computer Software Documentation," as such terms are used in 48 C.F.R. §12.212 or 48 C.F.R. §227.7202, as applicable. Consistent with 48 C.F.R. §12.212 or 48 C.F.R. §§227.7202-1 through 227.7202-4, as applicable, the Commercial Computer Software and Commercial Computer Software Documentation are being licensed to U.S. Government end users (a) only as Commercial Items and (b) with only those rights as are granted to all other end users pursuant to the terms and conditions herein. Unpublished-rights reserved under the copyright laws of the United States. Adobe Systems Incorporated, 345 Park Avenue, San Jose, CA 95110-2704, USA. For U.S. Government End Users, Adobe agrees to comply with all applicable equal opportunity laws including, if appropriate, the provisions of Executive Order 11246, as amended, Section 402 of the Vietnam Era Veterans Readjustment Assistance Act of 1974 (38 USC 4212), and Section 503 of the Rehabilitation Act of 1973, as amended, and the regulations at 41 CFR Parts 60-1 through 60-60, 60-250, and 60-741. The affirmative action clause and regulations contained in the preceding sentence shall be incorporated by reference.

Document Update Status

CS5.5	Added 2 chapters	Added "What's New in CS5.5" and "CS5.5 Porting Guide"; minor changes elsewhere. All content is current for CS5 and CS5.5.
-------	------------------	---

Contents

	Introduction	5
	Terminology	5
1	What's New in the InDesign CS5.5 Products SDK	6
	C++ IDE Requirements	6
	Porting Content	6
	Exporting PDF/VT file	6
	EPub and HTML export	6
	Articles	7
	Linked Stories	7
2	CS5.5 Porting Guide	8
	Main Changes	8
	EPub and HTML Export	8
3	What's New in the InDesign CS5 Products SDK	9
	C++ IDE Requirements	9
	Porting Content	9
	Scripting Events	9
	InDesign-Specific Creative Suite SDK Sample	10
	Multithreading	10
	Rich Interactive Documents	11
	Straddle Heads	11
	Multiple Page Sizes	12
	Middle Eastern Language Support	12
	Document Installed Fonts	13
4	CS5 Porting Guide	14
	Visual Studio 2008 Transition	14
	Setup	14
	Overview of project file changes	16
	Project conversion	18
	Update build events	20
	Xcode Changes	21
	Configuration files	21
	Update build events	21
	Main Changes	22
	Multithreading	22

Model and UI Separation	24
Multiple page sizes	25
Scripting changes	25
Fonts folder for Package	26
Resource folder	26
Resource APIs	26
Porting Recipes	27
AGMGraphicsContextWith	28
CDoClterationProvider	28
CHandleShape	28
CollectionEvent	28
CShape	28
GraphicsData	28
PluginVersion	28
ICallback	28
ICellAdornment	29
IDoClterationProvider	29
IGraphicsContext	29
IHandleShape	29
IAdornmentHandleShape	29
IAdornmentShape	29
IBaseSelectionHandlerData	29
IDrwEvtDispatcher	29
IDrwEvtHandler	30
IMasterPage	30
IMasterSpread	30
IMasterSpreadUtils	30
IINXManager	30
IOVERRIDEMasterPageItemData	31
IParcelShape	31
IPasteboard	31
IPasteboardPrefs	31
IPasteboardUtils	31
IShape	31
IScriptEventData	32
IScriptProvider	32
ISpread	32
IShowPageItem	32
kNonUniqueIDBasedObjectScriptElement	32
kUniqueIDBasedObjectScriptElement	32
K2::shared_ptr	33
k_noNO	33
k_sISL	33
metadata::DateTime	33
PictureWidget	33
ScriptInfo and ScriptElementIDs	33
SDK_DIR	33
SnapshotUtils and SnapshotUtilsEx	33
TextWrapRef	34
VersionedScriptElementInfo	34

Introduction

This document describes how to update your SDK plug-in code and development environments for the Adobe® InDesign® CS5 and CS5.5 family of applications. It details changes in the public API and other aspects of the SDK since the CS4 release. It contains the following:

- ▶ [Chapter 1, “What’s New in the InDesign CS5.5 Products SDK,”](#) summarizes the changes in CS5.5 since CS5.
- ▶ [Chapter 2, “CS5.5 Porting Guide,”](#) provides more details about changes to plug-in code and development environments for CS5.5, and helps you to port your plug-ins from CS5 to CS5.5.
- ▶ [Chapter 3, “What’s New in the InDesign CS5 Products SDK,”](#) summarizes the changes to the SDK from CS4 to CS5.
- ▶ [Chapter 4, “CS5 Porting Guide,”](#) provides more details about changes to plug-in code and development environments for CS5, and helps you to port your plug-ins from CS4 or earlier to CS5.

Terminology

<SDK> refers to the download location for the InDesign products SDK.

1 What's New in the InDesign CS5.5 Products SDK

This chapter summarizes the changes to the SDK in the CS5.5 release (since the CS5 release).

C++ IDE Requirements

InDesign CS5.5 uses the same development environment as CS5.

Porting Content

There are a few new public APIs in CS5.5. See <SDK>/docs/references/APIAdvisor_InDesign_CS5_vs_CS5.5.html for a diff between the CS5 and CS5.5 APIs.

Exporting PDF/VT file

InDesign CS5.5 introduces support for exporting PDF/VT files. PDF/VT is a PDF-based document-description format that is optimized for production of variable-data documents. The goal of this standard is to define a PDF-based format for describing variable-data documents in a way that will allow for faster, more efficient, and more consistent quality in the production of variable-data documents.

The InDesign CS5.5 Products SDK includes the following PDF/VT-related content:

- ▶ The "PDF Import and Export" chapter of the *Adobe InDesign Plug-in programming Guide Volume 2: Advanced Topics* contains information on exporting PDF/VT files.
- ▶ <SDK>/source/sdksamples/pdfvt and <SDK>/source/sdksamples/pdfvtui are two sample plug-ins to demonstrate the use of the new PDF/VT feature.

EPub and HTML export

The EPub export and HTML export (originally written using scripting) have been completely rewritten using C++, the native programming language in which core InDesign is written.

The InDesign CS5.5 Products SDK includes the following EPub and HTML export-related content:

- ▶ The "EPub Export" chapter of the *Adobe InDesign Plug-in programming Guide Volume 2: Advanced Topics* contains information on exporting EPub files.
- ▶ <SDK>/source/sdksamples/codesnippets/SnpExportEPub.cpp demonstrates how to use related APIs.
- ▶ <SDK>/scripting/indesign/scriptingguide/scripts/JavaScript/documents/ExportEPub.jsx
- ▶ <SDK>/scripting/indesign/scriptingguide/scripts/JavaScript/documents/ExportEPubWithOptions.jsx

Articles

InDesign CS5.5 introduces support for articles. Articles provide an easy way to create relationship among page items. These relationships can be used to define the content to be used while exporting to EPub, HTML, or Accessible PDFs; and to define the order of the content.

The InDesign CS5.5 Products SDK includes the following articles-related content:

- ▶ The “Articles” chapter of the *Adobe InDesign Plug-in Programming Guide Volume 2: Advanced Topics* contains information on using articles.
- ▶ `<SDK>/source/sdksamples/codesnippets/SnpManipulateArticles.cpp` demonstrates how to use related APIs.
- ▶ `<SDK>/scripting/indesign/scriptingguide/scripts/JavaScript/pageitems/AddRemoveArticleMember.jsx`
- ▶ `<SDK>/scripting/indesign/scriptingguide/scripts/JavaScript/pageitems/CreateArticle.jsx`
- ▶ `<SDK>/scripting/indesign/scriptingguide/scripts/JavaScript/pageitems/RemoveArticle.jsx`
- ▶ `<SDK>/scripting/indesign/scriptingguide/scripts/JavaScript/pageitems/ReorderArticleMembers.jsx`
- ▶ `<SDK>/scripting/indesign/scriptingguide/scripts/JavaScript/pageitems/ReorderArticles.jsx`

Linked Stories

InDesign CS5.5 introduces linked stories, which makes it easier to manage multiple versions of the same story or text content in the same document. Linked stories behave similarly to traditional links. You can designate a story as a parent, and then place the same story at other places in the document as child stories. Whenever you update the parent story, the child stories can be updated to synchronize with the parent story.

The InDesign CS5.5 Products SDK includes the following linked stories-related content:

- ▶ The “Linked Stories” chapter of the *Adobe InDesign Plug-in Programming Guide Volume 2: Advanced Topics* contains information on using linked stories.
- ▶ `<SDK>/source/sdksamples/codesnippets/SnpManipulateTextFrames.cpp` has been updated to demonstrate how to use related APIs.
- ▶ `<SDK>/scripting/indesign/scriptingguide/scripts/JavaScript/text/CreateLinkedStories.jsx`

2 CS5.5 Porting Guide

This chapter describes changes to plug-in code and development environments caused by changes in the public API and other aspects of the SDK for the InDesign CS5.5 family of applications since CS5, and helps developers port their plug-ins to CS5.5 from CS5.

Main Changes

EPub and HTML Export

In past releases, EPub and HTML export were written using scripting. In CS5.5, they are rewritten using C++ and the scripting APIs have been changed. Refer to `<SDK>/scripting/indesign/scriptingguide/scripts/JavaScript/documents/ExportEPub.jsx` and `<SDK>/scripting/indesign/scriptingguide/scripts/JavaScript/documents/ExportEPubWithOptions.jsx` to see how to use the new scripting APIs.

3 What's New in the InDesign CS5 Products SDK

For the benefit of previous users, this chapter summarizes the changes to the SDK in the CS5 release (since CS4), including:

- ▶ [“C++ IDE Requirements” on page 9](#)
- ▶ [“Porting Content” on page 9](#)
- ▶ [“Scripting Events” on page 9](#)
- ▶ [“InDesign-Specific Creative Suite SDK Sample” on page 10](#)
- ▶ [“Multithreading” on page 10](#)
- ▶ [“Rich Interactive Documents” on page 11](#)
- ▶ [“Straddle Heads” on page 11](#)
- ▶ [“Multiple Page Sizes” on page 12](#)
- ▶ [“Middle Eastern Language Support” on page 12](#)
- ▶ [“Document Installed Fonts” on page 13](#)

C++ IDE Requirements

On Windows®, the required C++ development environment is now Visual Studio 2008 with Service Pack 1 and Microsoft® Windows XP Service Pack 2 or higher.

On Mac OS®, the required C++ development environment is XCode 3.1.3 and Mac OS X 10.5.x.

Porting Content

Porting content is provided to help you transition to InDesign CS5. [Chapter 4, “CS5 Porting Guide,”](#) discusses porting procedures and the main changes to the InDesign API. The changes discussed include impacts of multithreading, multiple page sizes, script providers, and other minor changes.

See `<sdk>/docs/references/APIAdvisorID6_vs_ID7.html` for a *diff* between the CS4 and CS5 APIs.

Scripting Events

Scripting events were introduced in CS3 but have been greatly improved in CS5. They allow scripting code to be called when particular changes occur in the application or to a document. CS5 adds numerous new events that allow more powerful solutions to be implemented with scripting.

The new events fall into the following categories:

- ▶ Application shutdown
- ▶ Window open/close

- ▶ Active context changes
- ▶ Selection and selection attribute changes
- ▶ Idle tasks
- ▶ Place
- ▶ Links
- ▶ Tools

The SDK now provides support for plug-ins to add custom events.

The InDesign CS5 Products SDK includes the following related material:

- ▶ "Events" chapter in the *Adobe InDesign Scripting Guide*
- ▶ Events documentation scripts:
 - ▷ `<SDK>/scripting/indesign/scriptingguide/scripts/JavaScript/events`
- ▶ Events in script editor references (such as the OMV for ExtendScript Toolkit)
- ▶ Events in the scripting DOM dumps:
 - ▷ `<SDK>/docs/references/scripting-dom-applescript-idr70.html`
 - ▷ `<SDK>/docs/references/scripting-dom-javascript-idr70.html`
 - ▷ `<SDK>/docs/references/scripting-dom-visualbasic-idr70.html`
- ▶ "Feature Development with Scripting" chapter in *Getting Started with InDesign Plug-In Development*
- ▶ "Scripting Events" chapter in the *Adobe InDesign Plug-In Programming Guide*
- ▶ CustomScriptEvents plug-in

InDesign-Specific Creative Suite SDK Sample

The SDK includes an InDesign-specific Creative Suite SDK sample here:

```
<SDK>/source/sdksamples/flexuistroke
```

NOTE: This sample was rewritten in CS5 to make use of the Creative Suite SDK and InDesign CS5 selection script events.

Multithreading

InDesign CS5 contains some multithreaded features. This capability is not directly exposed to third parties, but it impacts porting. InDesign's multithreading support is model-based. This amounts to the ability to carry out model operations asynchronously on background threads. To support this, model/UI separation is mandatory in InDesign CS5. Model plug-ins must be made thread safe. UI plug-ins are executed only on the main thread.

The InDesign CS5 Products SDK includes the following content on the impact of multithreading:

- ▶ [“Multithreading” on page 22](#) in the [Chapter 4, “CS5 Porting Guide”](#)
- ▶ “Multithreading” chapter in the *Adobe InDesign Plug-in Programming Guide*
- ▶ “Model/UI Separation” chapter in the *Adobe InDesign Plug-in Programming Guide*
- ▶ Static/Global detection Tool: `<SDK>/devtools/statics_reporter`

NOTE: All model plug-in code is now thread safe.

Rich Interactive Documents

InDesign CS5 contains several new features and improvements that allow users to create interactive documents. These features, called *rich interactive documents* (RID), include the following:

- ▶ Animation — Page item position, size, and appearance can be altered over a period of time.
- ▶ Timing — Controls when and how animations are triggered. Animations can be ordered, grouped, and delayed. Timing also controls which events trigger the animations.
- ▶ Media support — InDesign supports a number of new media (audio and video) formats.
- ▶ Multistate objects — Multistate objects allow a single pageitem or a group of pageitems to be treated as a single state. States are analogous to layers in that their visibility can be turned on and off. However, only one state in a multistate object can be viewed at a time. There is no limit to the number of states that a multistate object can contain.

The InDesign CS5 Products SDK includes the following RID-related content:

- ▶ “Rich Interactive Documents” chapter in the *Adobe InDesign Plug-In Programming Guide*.
- ▶ A number of new code snippets that demonstrate RID capabilities. The following snippets are available in the `<SDK>/source/sdksamples/codesnippets` folder:
 - ▷ `SnExportDynamicDocument.cpp` — Updated to specify web intent and to use the `iAppearanceltemFacade` to create buttons.
 - ▷ `SnAddMediaFile.cpp` — Displays a media file and applies a controller skin to it.
 - ▷ `SnCreateCustomAnimation.cpp` — Exports an existing motion preset to a file or imports a new motion preset from a file.
 - ▷ `SnCreateAnimatedMultiStateObject.cpp` — Demonstrates how to use multistate objects, timings, and animations in C++.
- ▶ “Creating Dynamic Documents” chapter in the *Adobe InDesign Scripting Guide*.
- ▶ *Scripting Guide* scripts:
 - ▷ `<SDK>/scripting/indesign/scriptingguide/scripts/JavaScript/RID`

Straddle Heads

InDesign CS5 introduces straddle heads. A straddle head is a paragraph that stretches across multiple columns in a multicolumn text frame. A *full straddle* refers to a paragraph that stretches across all columns, while a *partial straddle* is a paragraph that spans more than a single column but less than all columns. In

addition to spanning columns, a straddling paragraph also gathers all preceding text and forces that text to push upward into equally balanced columns.

The InDesign CS5 Products SDK includes the following straddle head related content:

- ▶ The “Text Fundamentals” chapter of the *Adobe InDesign Plug-In Programming Guide* contains information on straddle heads.
- ▶ `<SDK>/scripting/indesign/scriptingguide/scripts/JavaScript/text/SpanColumns.jsx`

Multiple Page Sizes

InDesign CS5 introduces support for multiple page sizes in a single document. Earlier versions limited documents to a single page size. The multiple page size feature allows different sizes of pages within a single InDesign document. There are two primary use cases:

- ▶ The user wants to create a gate-fold page that is narrower than the standard pages in the document.
- ▶ The user wants a single InDesign document to contain a few different single pages of differing size. An example would be an Ad Campaign that contains a 3” by 5” postcard, 8 1/2” by 11” flyer, a web banner ad, and an 18” by 24” poster.

The InDesign CS5 Products SDK includes the following content related to multiple page sizes:

- ▶ Updates in the “Documents” chapter in the *Adobe InDesign Scripting Guide* that explain multiple page sizes.
- ▶ Scripts that alter page sizes:
 - ▷ `<SDK>/scripting/indesign/scriptingguide/scripts/JavaScript/documents/MasterPageTransform.jsx`
 - ▷ `<SDK>/scripting/indesign/scriptingguide/scripts/JavaScript/documents/PageReframe.jsx`
 - ▷ `<SDK>/scripting/indesign/scriptingguide/scripts/JavaScript/documents/PageResize.jsx`
 - ▷ `<SDK>/scripting/indesign/scriptingguide/scripts/JavaScript/documents/PageSelect.jsx`
 - ▷ `<SDK>/scripting/indesign/scriptingguide/scripts/JavaScript/documents/PageTransform.jsx`
- ▶ [“Multiple page sizes” on page 25.](#)
- ▶ `<SDK>/source/sdksamples/codesnippets/SnpManipulateSpreadsAndPages.cpp`

Middle Eastern Language Support

InDesign CS5 introduces Middle Eastern (ME) language support in the InDesign model. This includes scripting and API support in all InDesign CS5 products, but not user interface support. The license restricts partners from creating a localized user interface for these features. The ME features provide support for Arabic, Hebrew, and Greek. Arabic and Hebrew are significant in that they compose right to left.

InDesign CS5 includes the following ME-related content:

- ▶ ME-related content in the “Text Fundamentals” chapter in the *Adobe InDesign Plug-In Programming Guide*

- ▶ New chapter, "Middle Eastern Scripting Guide" in the *Adobe InDesign Server Scripting Guide* in the InDesign CS5 Server SDK

Document Installed Fonts

InDesign documents are often shared between computers that do not have the same fonts installed. The document installed fonts feature was introduced to avoid missing fonts. This feature allows users to package a document with the necessary fonts needed to share the document across installations of InDesign.

The InDesign CS5 Products SDK includes the following content related to document installed fonts:

- ▶ New information in the "Fonts" section of the *Adobe InDesign Plug-In Programming Guide*.
- ▶ ["Fonts folder for Package" on page 26](#).
- ▶ A code snippet that shows how to use related APIs:
 - ▷ `<SDK>/source/sdksamples/codesnippets/SnpInspectFontMgr.cpp`

4 CS5 Porting Guide

This chapter describes changes to plug-in code and development environments caused by changes in the public API and other aspects of the SDK for the InDesign CS5 family of applications since CS4, and helps developers port their plug-ins to CS5 from CS4.

Visual Studio 2008 Transition

Setup

Developing plug-ins for InDesign CS5 requires Microsoft Visual Studio 2008.

Installing Visual Studio

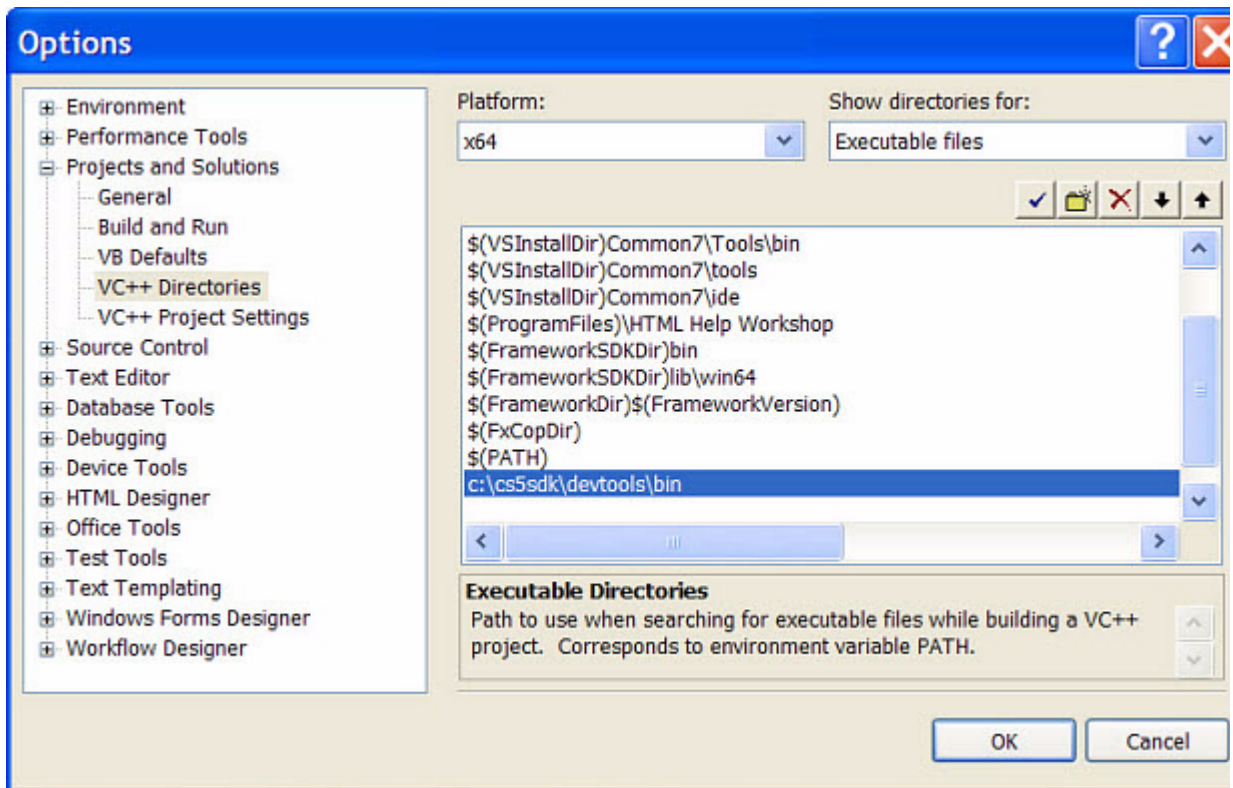
Setting up Visual Studio 2008 is fairly straightforward: Run the installer and select the custom install option. At a minimum, install the Visual C++ compiler and the x64 Compilers and Tools. Installing the x64 support enables you to build 64-bit versions of your plug-ins.

NOTE: 64-bit plug-ins are supported only by Adobe InDesign Server.

Specifying InDesign devtools for Visual C++

As with previous versions of Visual C++, you must configure Visual C++ to find InDesign-specific custom build tools, such as `schuss` ODFRC. The only real difference is that, in the past, you might not have configured the x64 platform. Now, we recommend that you configure both the Win32 and x64 platforms as follows:

1. Start Visual Studio 2008.
2. Select Tools > Options.
3. Expand the Projects and Solutions section and select VC++ Directories.
4. In the Show Directories For drop-down list, choose Executable Files.
5. In the Platform drop-down list, choose Win32.
6. Add a path to your local copy of the SDK's devtools/bin directory.
7. In the Platform drop-down list, choose x64 as shown in the following figure.



8. Once again, add a path to your local copy of the SDK's devtools/bin directory

Disabling IntelliSense

IntelliSense is the feature that implements auto-complete in Visual Studio. Some developers prefer to turn this feature off because it slows down the IDE and does not work particularly well with the InDesign code base.

To disable IntelliSense completely, remove or rename the following file:

```
C:\Program Files\Microsoft Visual Studio 9.0\VC\vcpackages\feacp.dll
```

You also can disable IntelliSense for a particular language; for example, to disable it for C/C++:

1. In Visual Studio, select Tools > Options.
2. Expand Text Editor.
3. Expand C/C++.
4. Select General.
5. Deselect Auto List Members in the Statement Completion section.

Enabling or disabling source-code control

You can enable or disable source control support using the Options dialog:

1. In Visual Studio, select Tools > Options.

2. Expand Source Control.
3. Select Plug-in Selection.
4. To disable source-control support, set this to None. To enable source-control support, select the plug-in for your source-control system. If it is installed, you will see a plug-in for your source-control system. You may need install or reinstall this component.

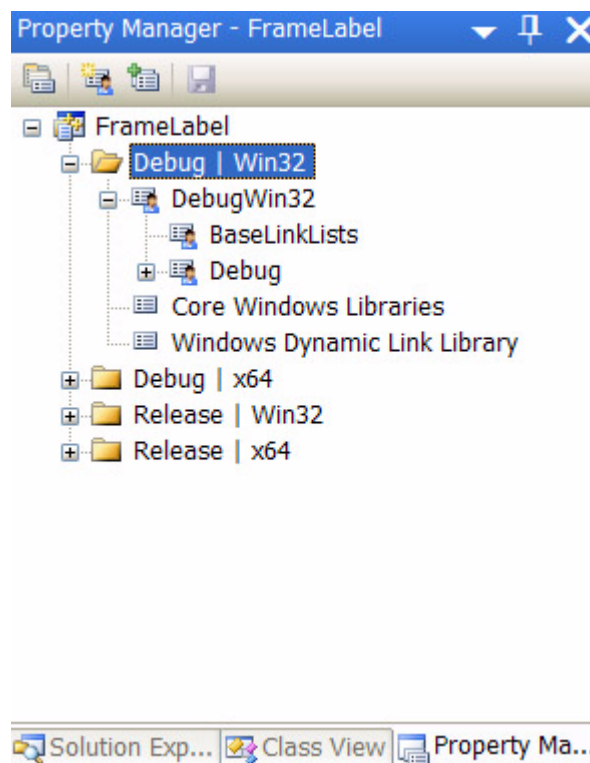
Overview of project file changes

There are modest changes to project files that accompany the move to Visual Studio 2008. Visual Studio project files (internal, open, samples, and DollyXs generated) now make use of property-sheet files (*.vsprops). These are XML files that contain the properties for a configuration or set of configurations. This simplifies project files and centralizes many project settings.

Configurations

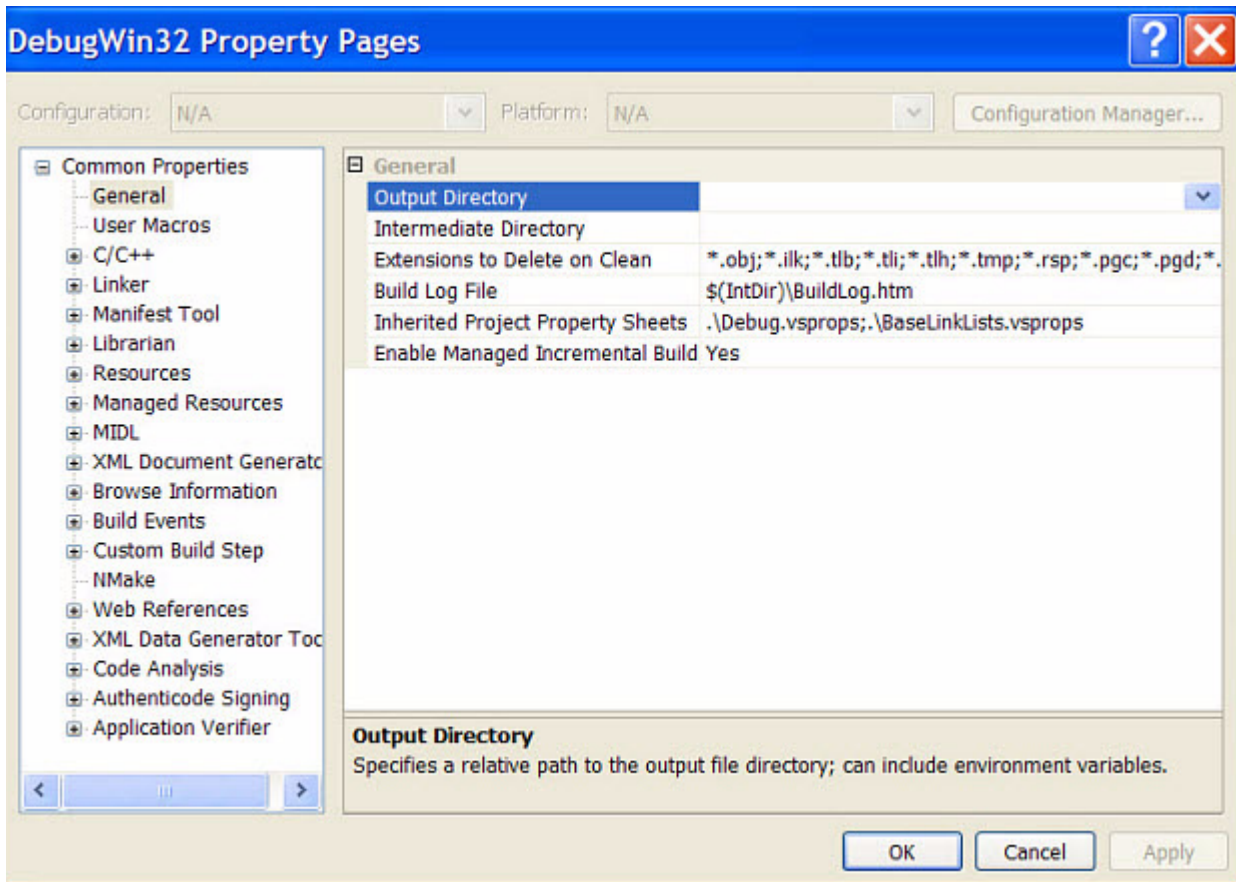
Each sample contains four configurations (as shown in the following Property Manager panel):

- ▶ Debug (also called Debug | Win32)
- ▶ Debug | x64
- ▶ Release (also called Release | Win32)
- ▶ Release | x64



Each configuration can specify a property sheet and its own overrides. Property sheets, in turn, can inherit from other property sheets. This is covered in more detail below.

You can inspect and edit configuration in the Property Manager panel. You can edit individual property sheets or apply overrides in the project. To edit a property sheet, double-click it in the Property Manager panel tree view (for example, DebugWin32) and apply changes in the subsequent dialog (as shown in the following figure). To apply overrides, double-click the configuration name (for example, Debug | Win32) and apply changes in the subsequent dialog.



It also is straightforward to inspect and edit project files in a text or XML editor. The following example shows the XML in a project file.

```
<?xml version="1.0" encoding="Windows-1252"?>
<VisualStudioProject
  ProjectType="Visual C++"
  Version="9.00"
  Name="FrameLabel"
  ProjectGUID="{EE97D690-6E66-45B8-B041-433D204A1C52}"
  RootNamespace="FrameLabel"
  TargetFrameworkVersion="0"
>
<Platforms>
  <Platform Name="Win32" />
  <Platform Name="x64" />
</Platforms>
<ToolFiles>
</ToolFiles>
<Configurations>
  <Configuration
    Name="Debug|Win32"
    OutputDirectory="..\objD\Framelabel"
    IntermediateDirectory="..\objD\Framelabel"
    ConfigurationType="2"
    InheritedPropertySheets="..\DebugWin32.vsprops"
  <!-- text omitted-->

```

Configuration property inheritance

In the preceding example, you can see that the project defines two platforms, Win32 and x64. You also can see one of the four configuration elements, Debug|Win32. This target inherits its properties from the DebugWin32.vsprops property sheet. In the XML file, this is specified in the InheritedPropertySheets attribute on this configuration's Configuration element. The InDesign SDK includes four property files that are used at this level:

- ▶ DebugWin32.vsprops
- ▶ ReleaseWin32.vsprops
- ▶ DebugX64.vsprops
- ▶ ReleaseX64.vsprops

These four property files inherit properties from Debug.vsprops or Release.vsprops files, which contain settings specific to the target (debug or release) but not the platform (Win32 or x64). In turn, Debug.vsprops and Release.vsprops inherit from Base.vsprops, which contains properties common to all configurations.

All these property sheets are parameterized with the ID_SDK_DIR macro. This identifies the path from the project to the SDK root. The four platform-specific property files covered above define this variable using a relative path from the <SDK>/build/win/prj folder to the root of the SDK. If your project file is not within the prj folder (or a sibling), you will need to override the ID_SDK_DIR macro.

Project conversion

There are two ways to approach project conversion. The first is to open the project in Visual Studio 2008 and allow the tool to convert the project. Because the formats are so similar, this really does not do much. For this reason, we recommend manually converting the projects in a text or XML editor.

Manual file project conversion

The following are the steps to manually convert a project. These instructions assume that you have basic familiarity with XML; for instance, they assume that you understand the terms *element* and *attribute*.

NOTE: Placing your project and source within the SDK folder structure (in the prj folder) simplifies some parts of the conversion; however, the following instructions cover configuring projects in the prj folder and those that build outside the SDK.

1. Make a backup copy of your project. You may need to refer to it during these steps.
2. The project files' version number is stored in the Version attribute of the VisualStudioProject element; this is near the top of the file. Change the content to 9.0:

```
Version="9.0"
```

3. Copy the Configuration element from an existing sample. If you are working on converting a model plug-in, use the Configuration element from FrameLabel.vcproj. If you are working on a user-interface plug-in, use the Configuration element from FrameLabelUI.vcproj. For your convenience, many of the paths used in these settings are parameterized using the ID_SDK_DIR macro.

You now have content in your project files that needs to be adjusted.

4. Adobe typically uses long (FrameLabel) and short (FrmLbl) names in our SDK projects. Dolly generates projects this way as well, so if you originally used Dolly to create your project, you probably have long and short names. Replace these strings with the correct values from your project.
5. If your project files are stored within the SDK prj folder, the paths to the Visual Studio property sheets should be correct; however, if your project is stored outside the SDK, you need to override the ID_SDK_DIR macro. To do this, create one new Visual Studio property sheet that defines the ID_SDK_DIR macro with the path from your project file to the SDK root. For example:

```
<?xml version="1.0" encoding="Windows-1252"?>
<VisualStudioPropertySheet
  ProjectType="Visual C++"
  Version="8.00"
  Name="DebugWin32"
  InheritedPropertySheets=""
>
  <UserMacro
    Name="ID_SDK_DIR"
    Value="c:\cs5sdk"
  />
</VisualStudioPropertySheet>
```

6. Specify the new property sheets as the last item in each Configuration element's InheritedPropertySheets attribute. For example:

```
</ToolFiles>
<Configurations>
  <Configuration
    Name="Debug|Win32"
    OutputDirectory="..\objD\Framelabel"
    IntermediateDirectory="..\objD\Framelabel"
    ConfigurationType="2"
    InheritedPropertySheets="..\DebugWin32.vsprops;..\SDKPath.vsprops"
  >
```

7. Linker settings are specified for each of the four targets. Search for the string "VCLinkerTool." It should appear four times, once for each configuration. Libraries are specified in the AdditionalDependencies attribute. Where possible, these paths are parameterized with the ID_SDK_DIR macro. Check your backup copy to see whether you need to link against any additional libraries. Use the ID_SDK_DIR macro for any paths that specify resources in the SDK.
8. Remove all VCWebDeploymentTool elements.
9. Two files in your project are built from files in the SDK. To ensure that VCPlugInHeaders.cpp and PlugInStatics.cpp are coming from files in the new SDK, when you update these paths, use the ID_SDK_DIR macro. For an example, see the FrameLabel.vcproj file.
10. Visual Studio allows you to further override configuration properties on individual files. Most files do not need to do this: simply inheriting properties is sufficient. However, there are three files that do require additional overrides. This is done using FileConfiguration elements within the File element:
 - ▷ Find the File element for the project's RC file. It should have four child FileConfiguration elements. If necessary, correct the paths in the AdditionalIncludeDirectories attributes. Where these paths point to folders in the SDK, use the ID_SDK_DIR macro.
 - ▷ Find the File element for the project's VCPlugInHeaders.cpp file. Remove the Optimization and DisableSpecificWarnings attributes in each FileConfiguration element.
 - ▷ Find the File element for the project's FR file. Your FR file requires a FileConfiguration element, but it should already be correct. Check that the paths look correct.
11. Unless you have a specific reason for modifying properties on a specific file, remove FileConfiguration elements from all other File elements. (Be careful not to disturb the RC, FR, and VCPlugInheaders.cpp files.)
12. Your project probably uses RSP files to build header search paths for both C++ and ODFRC compilations. If necessary, fix the path to the RSP files. Then, fix the paths within your RSP file. Note: You cannot use the \$(SDK_LIB) macro in these files.
13. Save the file and open it in Visual Studio 2008. If this all works, you are ready to tackle compilation issues. Before attempting compilation, read ["Main Changes" on page 22](#).

Update build events

1. Open the project with Visual Studio 2008.
2. Open the project Property Pages.
3. Select Configuration Properties > Build Events > Pre-Link Event.
4. Append the following line to the command line:


```
& del /f /s "$(TargetDir) ($(TargetName) Resources)\*.idrc & xcopy /E
  "$(IntDir)\*.idrc "$(TargetDir) ($(TargetName) Resources)\ " /Y
```
5. Repeat the preceding steps for every Configuration and Platform.

Xcode Changes

Development tools on Mac OS remain unchanged for CS5. Developing Mac OS plug-ins for InDesign CS5 still requires Xcode 3.1.3.

Configuration files

The structure of project and configuration files has changed. All configuration files (*.xcconfig) were moved to the following directory in the SDK:

```
<SDK>/build/mac/prj/_shared_build_settings
```

Using the configuration files

To keep up to date with the settings used by Adobe, we recommend that you use the configuration files included in the SDK. How you use them depends on how you have configured your projects. You have either added your project to the SDK (in the <SDK>/build/mac/prj directory) or you are building from outside the SDK. If you are building from within the SDK, you should be able to use the xcconfig files in the same way that the sample projects use them. If you are building from outside the SDK, you will need to complete an extra step to get projects to build. You have two choices:

- ▶ Copy the _shared_build_settings directory from the SDK to your project directory.
- ▶ In your project directory, create a symbolic link to the _shared_build_settings of the SDK:

```
cd <YourProjectDir>  
ln -s /mysdk/build/mac/prj/_shared_build_settings _shared_build_settings
```

Using the ID_SDK_ROOT variable

Our Xcode configuration files use the ID_SDK_ROOT variable to reference SDK resources. This variable must store an absolute or relative path to the root of the SDK. By default, this is the relative path from the prj folder to the SDK Root (../..). You can override this by creating your own xcconfig file that #includes plugin.sdk.xcconfig. Your xcconfig file should then provide the correct path from your plug-in to the SDK.

The #include directives in the config files do not understand variables like ID_SDK_ROOT. This is unfortunate, because they need to include other xcconfig files from the SDK. The SDK xcconfig files assume that there is a directory named _shared_build_settings next to the project directory, so you need to either copy the directory or make a symbolic link in your project directory, as described above.

NOTE: When you write your own xcconfig to override ID_SDK_ROOT, make sure that the path does not contain any space characters, because spaces could cause a build error.

Update build events

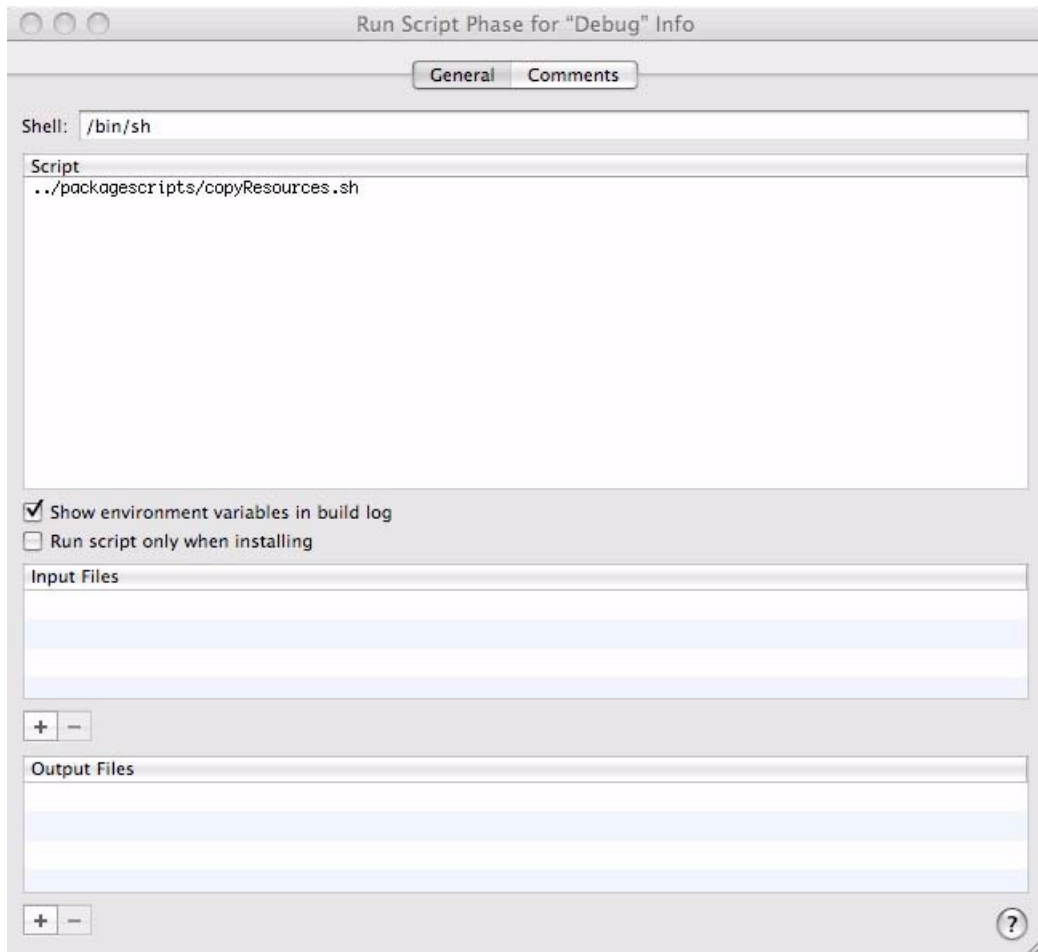
InDesign CS5 supports multithreaded resource access. As a result, the way ODFRC processes resource files (.fr files) also has changed. Several folders and files are copied to the resources directory while you build a plug-in.

A shell script, copyResources.sh, is provided in the SDK to copy all the resource files to the plug-in bundle. The full path name of the script is

```
<SDK>/build/mac/packagescripts/copyResources.sh
```

For a plug-in to be loaded by InDesign, the plug-in project must be updated as follows:

1. Create a new Run Script Build Phase for every target of your project. To do this, right-click a target and select Add > New Build Phase > New Run Script Build Phase.
2. Double-click the Run Script item under a target, to show the Run Script Phase screen:



3. Set Shell to /bin/shell.
4. Set Script to <SDK>/build/mac/packagescripts/copyResources.sh. You can specify this as a relative path if desired.

NOTE: If the path contains any spaces, it must be surrounded by double quotation marks.

Main Changes

Multithreading

In an effort to improve overall performance and responsiveness, Adobe made a significant investment in supporting multithreading in InDesign CS5. Multithreading in CS5 focuses on model operations, which yields the greatest impact. To do this, we invented a means for multiple threads to operate on an InDesign

database. This work required significant changes to the InDesign code base and probably affects your plug-in development.

IMPORTANT: You must ensure that your plug-ins are thread safe. If they are not, InDesign will behave inconsistently and may randomly crash.

NOTE: This section provides an *overview* of how to make your plug-ins thread safe. The *Adobe InDesign Plug-In Programming Guide* provides details about what multithreading affects and how to ensure that your plug-ins are thread safe.

Rules for thread safety

InDesign's multithreading environment provides a separate execution context (a cloned copy of its database) for each thread. Making a plug-in thread safe amounts to making all components safe to be instantiated and operated on by multiple threads simultaneously.

Threads do not share object-model instances. They do share globals and statics, so a big part of your effort is removing or synchronizing such variables.

The following rules summarize what plug-in developers must do to support multithreading:

- ▶ Separation of model and UI operations is now mandatory (see [“Model and UI Separation” on page 24](#)). A plug-in must declare whether it provides model support or user-interface (UI) support. For information on declaring model or UI support, refer to “Separating model and UI components” in the “Model and UI Separation” chapter of the *Plug-In Programming Guide*.
- ▶ All model plug-ins must be made thread safe. This amounts to eliminating or synchronizing global and external resources as described later in this chapter. (Because UI plug-ins are called only from the main thread, thread safety is not an issue.)
- ▶ Third-party code using its own background threads must not operate on InDesign object-model constructs from those threads.
- ▶ Service providers must now declare whether they are available on background threads. This is done using the `IK2ServiceProvider::GetThreadingPolicy()` method. You must call and implement service providers with this in mind. For details, refer to the “Threading and service providers” section in the *Programming Guide's* “Multithreading” chapter.
- ▶ Startup/shutdown services can now be called on application and/or thread start-up. This is specified using a `ServiceID`. For details, refer to the “Threading and startup/shutdown services” section in the *Programming Guide's* “Multithreading” chapter. Make sure that your service is using the appropriate ID.
- ▶ All libraries that are used by plug-in code must be thread safe. For example, InDesign uses the thread-safe versions of all Boost libraries.

Thread safety porting and development procedures

We recommend the following steps for porting plug-ins from prior versions of InDesign to support multithreading:

1. Read about “Thread-safe coding constructs” in the *Programming Guide's* “Multithreading” chapter.
2. If you have not already done so, enforce complete model/UI separation on all your plug-ins as described in “Separating model and UI components” in the “Model and UI Separation” chapter. Use the original prefix ID in the model plug-in, and acquire a new prefix ID for the UI components.

3. Update your `PlugInVersion` resource as described in “Separating model and UI components” in the “Model and UI Separation” chapter.
4. Do enough nonmultithreading CS5 porting to get your plug-ins compiling.
5. Review the “Threading and service providers” section in the *Programming Guide’s* “Multithreading” chapter and make sure that your code follows those rules. Note:
 - ▷ Ensure that your service providers are available on background threads if necessary.
 - ▷ Most significant service providers are available on background threads, but some service providers reside in a UI plug-in, hence are not available on background threads. Ensure that your model code does not rely on any UI service providers, or ensure that these service providers are moved to model plug-ins.
6. Review the “Threading and startup/shutdown services” section in the *Programming Guide’s* “Multithreading” chapter and make sure that your code meets the requirements.
 - ▷ Decide whether your startup/shutdown services need to be called on application and/or thread startup and shutdown. Use the appropriate service-provider implementation. In most cases, your code will be application specific, so you will most often use `kAppStartupShutdownProviderImpl`.
7. Run the Statics Reporter tool on your plug-ins. This tool detects and reports the use of any global or static variables in your plug-in.
8. Remove any globals and statics that you can. Some variables are acceptable: statically initialized const variables are fundamentally safe, but other global variables that change state are not safe. Such variables can be used to optimize performance, but they often are used just for convenience; often, such code can be efficiently rewritten without a global variable. Regardless, the fact that such code can run on multiple threads fundamentally changes the execution scenario.
9. Remove any function-local static variables. Substitute globals in an anonymous namespace with appropriate synchronization, or use nonstatic const objects.
10. Certain globals will be difficult or impossible to remove, so some synchronization will be required. Study the threading APIs and libraries, and use the appropriate constructs to make your code thread-safe. For your convenience, we include several thread safety recipes; in the *Programming Guide’s* “Multithreading” chapter.
11. If necessary, apply the same synchronization techniques to your use of external resources, such as files. For example, if your plug-in writes to an external text file, use the described synchronization strategies to ensure that only one thread writes to it at a time.
12. Launch the debug version of InDesign with your plug-in present. Fix any asserts or error messages related to multithreading. In particular, fix violations of object-model rules reported on startup, as described in the “Object-Model Rules” section in the *Programming Guide’s* “Multithreading” chapter.
13. Follow the guidance in the “Testing for thread safety” section in the *Programming Guide’s* “Multithreading” chapter.

Model and UI Separation

To support multithreading, the model and user interface (UI) components of several InCopy-related plug-ins have been separated into different plug-ins. These plug-ins had not previously been separated because they were not required by InDesign Server. The model component retains the original plug-in ID, and UI-related IDs were moved into the following new ID.h files:

- ▶ WritingModeUIID.h (These are the IDs for the Galley UI plug-in.)
- ▶ EditorHelperUIID.h
- ▶ InCopyFileActionsUIID.h
- ▶ AssignmentUIID.h
- ▶ InCopyCoreUIID.h

Wherever you are using a UI ID that has moved, you must include the new ID file.

Multiple page sizes

In past releases, InDesign has supported uniform page sizes. In CS5, it now supports multiple page sizes. To implement this, several related APIs were changed, including `INewSpreadCmdData` and `IApplyMasterCmdData`.

Refer to the API Advisor (`<SDK>/docs/references/APIAdvisorID6_vs_ID7.html`) for details. SDK code snippet `SnManipulateSpreadsAndPages` has been updated accordingly.

Scripting changes

Plug-ins that implement scripting require several changes. We changed some of our terminology to support new and improved capabilities in CS5. In the past, we overloaded the term *event*. It meant both the point you can register for notification (attachable events) and the methods you can call on scripting objects. CS5 corrects this confusion. The documentation and code will use *event* to mean attachable events and *method* to refer to callable subroutines that exist on a scripting object. The term *script request* is used in more generic contexts.

In your code, fix the following:

- ▶ Change the `IScriptEventData` to `IScriptRequestData`.
- ▶ Many of the method names in `IScriptRequest` also have changed.; for example, `ExtractEventData()` is now `ExtractRequestData()`. Simply substituting `Request` for `Event` is a good first guess.
- ▶ Other code names replace “event” with “method”; for example, `HandleEventOnObjects` in `IScriptProvider` is now `HandleMethodOnObjects`; `HandleEvent` in `CScriptProvider` is now `HandleMethod`.

NOTE: For a complete list of name changes, see [“IScriptEventData” on page 32](#), [“IScriptProvider” on page 32](#), and [“ScriptInfo and ScriptElementIDs” on page 33](#).

The terminology for script objects that can exist in a collection also has changed. Previously, we used the term *plural*; these are now referred to as *collection objects*. Build errors scripting symbols that contain “Plural” probably can be changed to “Collection.” For example, scripting-object resources used to use `NoPluralInfo` to indicate that they were not a collection; this is now `NoCollectionInfo`.

InDesign CS5 contains numerous new scripting features and this work affects numerous APIs. Refer to the API Advisor (`<SDK>/docs/references/APIAdvisorID6_vs_ID7.html`) for details.

Fonts folder for Package

The name of the fonts folder for the Package facility has changed from “Fonts” to “Document fonts.” In InDesign CS5, the fonts in the “Document fonts” folder will be loaded only while the document in the same directory is open. We call these fonts *document installed fonts*.

Resource folder

InDesign supports multithreaded resource access. As a result, the way ODFRC processes resource files (.fr files) also has changed. When a plug-in is compiled, ODFRC generates several new folders containing resource files. On Windows, these folders are in the “(<PluginName> Resources)” directory, parallel to the plug-in file. (Note that the parentheses are part of the directory name.) On Mac OS, the folders are in the Resources directory of the plug-in bundle. On Windows and Mac OS, if the folder name is inconsistent with the plug-in name or a resource file is missing, the plug-in cannot be loaded.

Resource APIs

CS4 and older code that looked up resources used the ResourceEnabler and API from the PlatformResourceShell namespace, which are not thread safe. In CS5, ODFRC and the way InDesign reads ODFRC resources have changed, so any resource that is compiled by ODFRC must be obtained using a new API in <SDK>/source/public/includes/IDResourceShell.h. Here is a quick look at how the new API compares with the old API using two of the most popular use cases.

Use Case 1: Single resource needs to be obtained and used

CS4 and older code:

```
ResourceEnabler resSetter(kMyPluginID);
SysHandle resource = PlatformResourceShell::GetResourceLI(RsrcSpec(locale,
kInvalidPlugin, kSomeRsrcType, kMyRsrcID));
if (strRsrc != nil)
{
    int32 dataSize = PlatformResourceShell::GetSiz eofResource(resource);
    uint8* buffer = *(uint8**)resource;
    //read the buffer using a stream and use it
}
```

PlatformResourceShell::ReleaseResource(resource);

CS5 code now looks like:

```
IDResource resource = IDResourceShell::GetLocalized(RsrcSpec(locale,
kMyPluginID, kSomeRsrcType, kMyRsrcID));
if (resource.IsValid())
{
    void* buffer = resource.Get();
    size_t resourceSize = resource.Length();
    //read the buffer using a stream and use it
    //we use something like:
    //InterfacePtr<IPMStream>
    resourceStream(StreamUtil::CreatePointerStreamRead((char*)buffer,
resourceSize));}
```

As you can see, the ResourceEnabler is not needed to load the plug-in’s resource file, and ReleaseResource is not needed. IDResource’s buffer is managed as described in IDResourceShell.h so that the client does not need to worry about memory management.

Use Case #2: A common operation needs to be performed on all resources in MyPlugin that are of kMyRsrcType

CS4 and older code:

```
ResourceEnabler resEnabler(kMyPluginID);
int16 resourceCount =
    PlatformResourceShell::CountResourcesLI(RsrcSpec(kInvalidPlugin,
    kMyRsrcType, kFalse, 0));
for (int16 i = 1; i <= count; ++i)
{
    SysHandle resource =
        PlatformResourceShell::GetResourceLI(RsrcSpec(kInvalidPlugin,
        kFileTypeTableRsrcType, i, kFalse));
    //using "i" in the RsrcSpec magically used to work since resources
    //that needed to be enumerated were named 1, 2, 3, ...
    //do the common operation on the resource
    PlatformResourceShell::ReleaseResource(resource);
}
```

CS5 code now looks like:

```
IDResourceShell::ResourceEnumerationCallback enumeration_callback =
boost::bind(&MyCallbackFunction, boost::ref(this), _1);
IDResourceShell::EnumerateResourcesByType(kMyPluginID, kMyRsrcType,
enumeration_callback);
```

And MyCallbackFunction looks something like:

```
bool MyClass::MyCallbackFunction(IDResource resource
/*, and maybe some optional parameters you'd like to use described below */) {
    //do the common operation on the resource
    return true; //to continue enumerating
}
```

The callback is actually:

```
typedef boost::function<bool (IDResource r, void* userData, RsrcSpec spec)>
ResourceEnumerationCallback;
```

So you can pass in some user data if you want to, or examine the RsrcSpec. No more resource counting, no more loops, no more needing to know that the resources are somehow named 1, 2, 3... The callback function is called for each kMyRsrcTypes in MyPlugin.

Note that there may be some resources that are not complied with ODFRC. These might be anything, such as old Mac cursors in an .rsrc file somewhere. This is not a recommended practice. Nevertheless, if you have some of these, you will need to keep using ResourceEnabler and PlatformResourceShell to read those resources.

Porting Recipes

This section includes a handful of changes that you will likely run into right away. This may help you get beyond the most common compilation errors quickly.

AGMGraphicsContextWith

See [“IGraphicsContext” on page 29](#).

CDoClterationProvider

See [“IDoClterationProvider” on page 29](#).

CHandleShape

`DrawAdornment()` was changed to `DrawAdornments()`, because multiple handle-shape adornments can be present. This is only a name change.

Also see [“IHandleShape” on page 29](#).

CollectionEvent

The `CollectionEvent{ kCountEventScriptElement }` directive is now unnecessary. This used to be required to add a “count” method to collections of children on an object. Because this method is required for all collections, we added code to the scripting DOM constructor to add it automatically; therefore, it is no longer necessary to add this directive in scripting resources (though there is no functional harm in doing so).

CShape

The first argument of `IterateDrawOrderHierarchy` is now `const`. This should never have been changeable. If your code changes this `PMMatrix`, it is in error.

See [“IShape” on page 31](#).

GraphicsData

See [“IGraphicsContext” on page 29](#).

PluginVersion

In InDesign CS5, a plug-in must identify itself as model or UI; it can no longer be both. The plug-in type is identified in the plug-in’s `PluginVersion` resource in its FR file. If you do not fix this, you will get an error in the `PluginVersion` resource when ODFRC tries to compile your resource file. To fix this, `#include PluginModel_UIAttributes.h` in your FR file and add a line (before the version and after the feature set) that contains `kUIPlugIn` or `kModelPlugIn`. For an example, see the sample projects.

ICallback

Change `Callback` to `ExecuteCallback`.

Change `GetDrawManager` to `GetCallbackDrawManager`.

ICellAdornment

Change IterateDrawOrder to IterateCellAdornmentDrawOrder.

IDocIterationProvider

The first argument of VisitPageItem is now const. This should never have been changeable. If your code changes this PMMatrix, it is in error.

IGraphicsContext

Change GetInverseTransform to GetViewToContentTransform.

Change GetTransform to GetContentToViewTransform

IHandleShape

Change Draw to DrawHandleShape.

Change GetPaintedBBox to GetPaintedHandleBounds.

IAdornmentHandleShape

Change Draw to DrawAdornmentHandleShape.

Change GetPaintedBBox to GetPaintedAdornmentHandleBounds.

IAdornmentShape

Change Draw to DrawAdornment.

Change GetPaintedBBox to GetPaintedAdornmentBounds.

IBaseSelectionHandlerData

Change GetRect to GetHitTestRectangle .

Change Set to SetHitTestRectangle.

IDrwEvtDispatcher

Overloaded ProcessEvent methods were changed to new descriptive method names. Refer to the API Advisor ([SDK](http://<SDK>/docs/references/APIAdvisorID6_vs_ID7.html)/docs/references/APIAdvisorID6_vs_ID7.html) for details.

- ▶ ProcessDrawEvent
- ▶ ProcessDrawFilterEvent
- ▶ ProcessFilterEvent
- ▶ ProcessHitTestEvent

- ▶ ProcessHitTestFilterEvent
- ▶ ProcessInvalEvent
- ▶ ProcessInvalFilterEvent
- ▶ ProcessIterateEvent

IDrwEvtHandler

Change `HandleEvent` to `HandleDrawEvent`.

IMasterPage

Change `GetMasterIndex` to `GetMasterSpreadPageIndex`.

Change `GetMasterPageUID` to `GetMasterSpreadUID`.

Change `SetMasterIndex` and `SetMasterPageUID` to `SetMasterPageData`, which combines both operations into one method.

IMasterSpread

Change `GetCurrentViewOffset` to `GetCurrentSpreadOffset`. The new method returns a `PMatrix` reference instead of a `PMPoint`. Master pages used to need to be translated horizontally when mapped to an actual spread. Now they can be transformed in any way (rotated, scaled, and so on).

Change `SetCurrentViewOffset` to `SetCurrentSpreadOffset`. Similarly, the new method takes a `PMatrix` where it used to take a `PMPoint`.

Change `GetDrawingPageUID` to `GetTopLevelPageUID`.

IMasterSpreadUtils

`AppendMasterPageItems` takes a `PMatrixCollection` instead of a `PMPointList`.

IINXManager

Clients (third-party developers) are now asked to register a callback function, which is then invoked for each threaded export to IDML. Clients are required to create an instance of `INXErrorHandler` and return it in their callback's implementation.

Public Interface changes:

- ▶ Changed prototype of `IINXManager::SetINXErrorHandler` to allow clients to register a callback.
- ▶ Changed prototype of `IINXManager::GetINXErrorHandler` to allow clients to retrieve the registered callback.

Two SDK samples changed to make use of these APIs:

- ▶ `<SDK>/source/sdksamples/inxerrorlogging`

- ▶ `<SDK>/source/sdksamples/inxerrorloggingui`

IOVERRIDEMASTERPAGEITEMDATA

Change the use of `PMPointList` to `PMMatrixCollection` in `GetItemShifts`, `GetNthOverride`, and `SetNthOverride`.

Change `Set` to `SetOverrideMasterPageItemData`.

IPARCELSHAPE

Change `IterateDrawOrder` to `IterateParcelShapeDrawOrder`.

IPASTEBOARD

Remove these two interface files:

- ▶ `<SDK>/source/public/interfaces/layout/IResizePasteboardCmdData.h`
- ▶ `<SDK>/source/public/interfaces/layout/IPasteboard.h`

Command `kResizePasteboardCmdBoss` was removed.

The `IPasteboard` interface was used to control `Get/SetPasteboardBounds` and the border around the pasteboard, but the border is now controlled in `IPasteboardPrefs.h`. As an aside, the bounds of a pasteboard can now be calculated on a per-spread basis like this:

```
InterfacePtr<IGeometry> spreadGeo(spreadList->QueryNthSpread(mySpreadIndex));
PMRect spreadBounds =
    Utils<Facade::IGeometryFacade>()->GetItemBounds(GetUIDRef(spreadGeo),
    Transform::PasteboardCoordinates(), Geometry::OuterStrokeBounds());
```

IPASTEBOARDPREFS

`GetPasteboardBorder` now takes two out parameters, the size of the horizontal and vertical borders around spreads on the pasteboard.

IPASTEBOARDUTILS

All three `GetPasteboardBounds` methods were removed. Now use the `Facade::IGeometryFacade`.

ISHAPE

The `Draw` method was replaced by `ProcessDrawShape`. The behavior of the new method is slightly different: The `Draw` method expected implementations to set up the graphics-port transform by calling `SetInnerToParentTransform(gPort)`, while `ProcessDrawShape` expects callers to handle this transformation.

Change `IterateDrawOrder` to `IterateShapeDrawOrder`. Callers used to pass a matrix, but it had to be the identity matrix for this to work correctly. The new method name removes the need to pass the identity matrix. An underlying method, `IterateShapeDrawOrder_`, still requires the matrix.

IScriptEventData

To avoid confusion created by the overloaded use of the term “event” in scripting APIs, the `IScriptEventData` class was changed to `IScriptRequestData`. Many uses of “event” were changed to “method or “request.” For more information, see [“Scripting changes” on page 25](#).

- ▶ Replace `IID_ISCRIPTEVENTDATA` with `IID_ISCRIPTREQUESTDATA`.
- ▶ Replace `InsertEventData` with `InsertRequestData`.
- ▶ Replace `ExtractEventData` with `ExtractRequestData`.
- ▶ Replace `HasEventData` with `HasRequestData`.
- ▶ Replace `SetEvent` with `SetMethod`.
- ▶ Replace `IsEvent` with `IsMethod`.
- ▶ Replace `SetEventWithProperties` with `SetMethodWithProperties`.
- ▶ Replace `IsEventWithProperties` with `IsMethodWithProperties`.

IScriptProvider

The term “Event” is now used only for attachable script events. Some uses were changed to “Method”:

- ▶ Replace `HandleEventOnObjects` with `HandleMethodOnObjects`.
- ▶ Replace `HandleEvent` with `HandleMethod`.

ISpread

The `coordinateSpace` parameter in `GetPagesAndItemsBounds()` is no longer optional, so the order of parameters changed.

IShowPageItem

`IShowPageItem` was replaced by `IShowPageItemUtils`. Its meaning and behavior were not changed.

kNonUniqueIDBasedObjectScriptElement

`kNonUniqueIDBasedObjectScriptElement` was changed to `kIDBasedObjectScriptElement`. The meaning and behavior were not changed. This always meant that the object has an ID, but it is not an InDesign UID. This name change clarifies the meaning.

kUniqueIDBasedObjectScriptElement

`kUniqueIDBasedObjectScriptElement` was changed to `kUIDBasedObjectScriptElement`. The meaning and behavior were not changed. This name change clarifies the fact that these are UID-based objects.

K2::shared_ptr

K2::shared_ptr is no longer available. Use boost::shared_ptr instead. This change was made for thread safety: K2::shared_ptr was not thread safe, while boost::shared_ptr is.

k_noNO

The locale symbol for Norway has changed from k_noNO to k_nbNO.

k_sISL

The locale symbol for Slovenian has changed from k_sISL to k_sSI.

metadata::DateTime

The metadata::DateTime struct was changed to use the XMP_DateTime struct supported by the XMP Toolkit. The toolkit's `<SDK>/external/bibxmp/include/XMP_Const.h` file needs to be included now, to get XMP_DateTime. This affects methods in the following interfaces:

- ▶ IAdobeCoreMetaData
- ▶ IMetaDataAccess
- ▶ ISetAdobeCoreMDCmdData

Developers who want to compare date-times in client code should consider using the TXMPUtils::CompareDateTime, found in `<SDK>/external/bibxmp/include/TXMPUtils.hpp`.

PictureWidget

PictureWidget and kPictureWidgetBoss were removed. Convert your images to PNG and use IconSuiteWidget. There are many examples of this in the sample plug-ins.

ScriptInfo and ScriptElementIDs

Replace EventScriptElement with MethodScriptElement.

SDK_DIR

Replace the SDK_DIR macro with ID_SDK_DIR.

SnapshotUtils and SnapshotUtilsEx

In order to make all clients of SnapshotUtils and SnapshotUtilsEx work simultaneously, an additional parameter is added to the constructors that take a clippingBounds parameter:

```
const PMMatrix & boundsToSpreadMatrix
```

The parameter is a PMMatrix that specifies the coordinate system of the clippingBounds rect. In particular, passing the specified rect through the matrix brings the rect into spread coordinates.

Existing callers that provided a clippingBounds in spread coordinates can simply pass the identity matrix PMMatrix() as the second parameter. Callers that compute a clippingBounds in some other coordinate space need to either rework their bounds calculations in spread coordinates or pass the matrix relating their bounds coordinates to spread coordinates.

These files are changed accordingly:

- ▶ `<SDK>/source/sdksamples/snapshot/SnapSelectionSuiteCSB.cpp`
- ▶ `<SDK>/source/sdksamples/snapshot/SnapTracker.cpp`
- ▶ `<SDK>/source/sdksamples/transparencyeffect/TranFxMatteFactory.cpp`

TextWrapRef

In struct TextWrapRef, change the two PMReal parameters fXOffset and fYOffset with the PMMatrix parameter fTransform.

Before the change, x and y offsets were stored for translating the master page wraps onto layout pages. As part of this change, the offsets are replaced by a matrix so wraps can work with scaled and rotated master pages. See `<SDK>/source/public/includes/TextWrapRef.h`.

VersionedScriptElementInfo

There are many name changes in items that can appear in VersionedScriptElementInfo:

- ▶ Replace occurrences of Event with Method.
- ▶ Replace ObsoleteEvent with ObsoleteMethod.
- ▶ Replace CollectionEvent with CollectionMethod.
- ▶ Replace NoPluralInfo with NoCollectionInfo.