Xcode Tools Sensei Changes for Xcode 2.4

This document lists the changes I've made from the previous version of Xcode Tools Sensei to the latest version. Most of the changes are due to updates in the Xcode Tools, but I also added material that needed to be in the book. People who bought an older version of the book will benefit the most from this document, but there is some useful information for other Mac OS X developers.

This document lists the changes from Xcode 2.2 to Xcode 2.4. I have a document on the Xcode Tools Sensei site that details the changes I made from Xcode 2.1 to Xcode 2.2.

I've organized the changes by chapter and by section in each chapter. The section order reflects the order the sections appear in the book.

Chapter 1

Application Projects

Apple stopped adding features to the Java version of Cocoa in Mac OS X 10.4 so I would advise against writing new Cocoa applications in Java. Improvements Apple makes to Cocoa will not make their way into the Java version.

Creating Your Own Project Templates

The previous material on creating project templates had one flaw. If you created a project and used it as a template, the product name would be the name of the template, not the name of the project you created from the template. A better solution is to make a copy of one of Xcode's project templates, and make changes to that copy.

1) Go to `/Library/Application Support/Apple/Developer Tools/Project Templates`. This folder contains all the Xcode project templates.
2) Select the project type that most closely matches the template you want to create. Make a copy of that project type's folder. This copy is your template folder.
3) The project folder's name is what appears in the Project Assistant window. Change the name of your template folder.
4) Open the project file that resides in your template folder.
5) Make the changes you need to make to the project. Add frameworks and files. Add code to a source code file. Do whatever you have to do to make the project a suitable template.

Now when you create a new project, you should see your project template as one of the project choices in the Project Assistant window. There is one thing missing. When you select your project template, the description of the project you copied appears. How do you change the description?

The description appears in the project's `TemplateInfo.plist` property list file. This file resides in the project file's bundle. Control-click the project file in the Finder to open a contextual menu. Choose Show Package Contents from the menu to open a Finder window that shows the bundle's contents. You should see the `TemplateInfo.plist` file in the newly opened window.

Double-clicking the `TemplateInfo.plist` file opens it in the Property List Editor application. Click the disclosure triangle next to Root. You should see a property named Description. The Description property's contents is what you will see when you select the project from the project list when you create a new project. Double-click the Value column for the Description property and type the project's description.

Save the `TemplateInfo.plist` file, and your description should now appear when you create a new Xcode project. If the description does not appear, you may have to relaunch Xcode.

Adding Files You've Already Created

I added some explanation on when to recursively create groups and when to create folder references when adding a folder of files to your project. Recursively create groups when you need to access the individual files of the folder in Xcode. If you're adding a folder of source code files, you want to recursively create the group because you want to be able to examine and edit the source code files.

Create a folder reference when you don't need to access the individual files in Xcode. If you're writing a game and you have a folder named Sound Effects that holds the game's sound effects, create a folder reference when adding the Sound Effects folder to your project. You're not going to be doing anything with the sound files in Xcode. All you want to do is copy the folder to the application bundle when you build the project, and creating a folder reference lets you accomplish this task.

Configuration Settings Files

This section is new material. Xcode 2.1 introduced configuration files. A configuration file is a text file that contains build settings. If you find yourself changing the same build settings for all your projects, you should use a configuration file. Add the settings you're always changing to the configuration file. Add the configuration file to your project and tell Xcode to base the build on the configuration file. Xcode will use the settings in the configuration file to build your project so you don't have to change the settings in the build configuration information panel.

Creating a Configuration File

Because a configuration file is just a text file, you can create one in any text editor. Give the file the extension `.xcconfig` so Xcode knows the file is a configuration file. The easiest way to create a configuration file is to create it in Xcode. Choose File > New File to create a new file. Select Configuration Settings File from the list of file types. If your version of Xcode does not have the Configuration Settings File, select Empty File in Project, and make sure you give the file the extension `.xcconfig`.

When you create a new file, Xcode sets things up so the file is added to your project's targets. You do not want to add configuration files to targets. Deselect the checkbox next to each target in your project before clicking the Finish button to create the file.

Creating a configuration file in Xcode is easy, but you're not going to want to create a new configuration file for each project you create. The point of using a configuration file is to create a list of build settings once and use that list in multiple projects. Choose Project > Add to Project to add your configuration file to other projects. Make sure you do not add the configuration file to the other projects' targets.

Xcode projects can contain multiple configuration files. You can have one configuration file of debug build settings, a second configuration file of release build settings and use both files in your project. Use the debug configuration file in your debug build configuration, and use the release configuration file in your release build configuration.

What Goes in a Configuration File?

A configuration file contains a list of build settings, with one setting per line. Each setting takes the following form:

```
SETTING = value;
```

Xcode has lots of build settings, too many for me to list here. Choosing Help > Show Build Settings Notes provides an overview of Xcode's build settings. Selecting a build setting in Xcode's build configuration information panel shows the formal build setting name in brackets. Build settings usually contain all uppercase letters. If you select the Optimization Level build setting, which is part of the Code Generation build settings collection, you will see the setting's formal name is `GCC_OPTIMIZATION_LEVEL`.

Suppose you want all of your projects to use the DWARF debugging format. You would add the following setting to the configuration file:

```
DEBUG_INFORMATION_FORMAT = dwarf;
```

Xcode 2.2 introduced the ability to add comments to a configuration file. Xcode uses the C++ // comment style.

```
// This is a comment.
```

A configuration file can contain as many build settings as you want, but don't put every build setting in the configuration file. Put only the build settings you don't want to be constantly changing. If there are only three build settings you're constantly changing, put those three settings in the configuration file. Xcode will use the settings in the build configuration information panel for the rest of the build settings.

Telling Your Project to Use a Configuration File

After writing your build configuration files, you must add them to your project and tell Xcode what configuration file to use. If you haven't added configuration files to your project, choose Project > Add to Project to add them, making sure you don't add them to your project's targets.

To assign a configuration setting file, open the build configuration information panel for your project. In the lower third of the panel is the Based on pop-up menu. The menu contains the configuration files you added to your project. Choose a configuration file from the menu. Make sure you choose a configuration file for each build configuration in your project, assuming you want to use a configuration file for all your project's build configurations.

Overriding the Configuration File

Suppose you have a configuration file with 15 settings. You create a new project, but want to use only 13 out of the 15 settings in the configuration file. How do you tell Xcode to use only the 13 settings you want to use?

The solution is to use the build configuration information panel. When you change a build setting from the build configuration information panel, it overrides the setting in the configuration file. In the example from the last paragraph, you would open your project's build configuration information panel and change the two settings you want to override.

Because the build configuration information panel overrides the configuration file, be careful when you open the build configuration information panel. You don't want to accidentally override any of your configuration file's build settings.

Distributed Builds

Xcode 2.3 introduced dedicated network builds. Dedicated network builds are designed to build projects using 12 or more Macs so most of you won't be able to take advantage of them. If you have enough Macs to use dedicated networked builds, choose Dedicated network build from the menu next to the Distribute via checkbox in the distributed builds preferences.

Building 64-bit Universal Binaries

Xcode 2.4 added the ability to create 64-bit Intel binaries. Add the `x86_64` architecture to the Architectures build setting to create a 64-bit binary. If you add the `ppc64` architecture, Xcode will create a 64-bit binary for G5 Macs.

Most of Apple's frameworks, including the Cocoa and Carbon frameworks, do not have 64-bit support so there is no reason for most of you to create 64-bit binaries. If you try to compile a 64-bit version of a Cocoa application on Mac OS X 10.4, you will get a bunch of compiler errors. Unix scientific applications that need to do a lot of number crunching on Mac Pros or G5 Macs are the applications that would benefit most from 64-bit binaries on Mac OS X 10.4. 64-bit support is coming in Mac OS X 10.5 so there are benefits to knowing how to build 64-bit binaries.

Chapter 2

Choosing a Debugging Format

Prior to Xcode 2.3 the Xcode debugger supported only the Stabs format. Xcode 2.3 added support for the DWARF debugging format. DWARF is a popular debugging format in the Unix world. It is more descriptive and flexible than Stabs. DWARF is also more efficient so less disk space is needed to store debugging information.

Although Xcode 2.3 added DWARF support, all Xcode projects except the Carbon C++ application projects use Stabs as the initial debugging format. To use DWARF as your debugging format:

1) Select your project name from the Groups and Files list.
2) Click the Info button to open the project's information panel.
3) Click the Build tab.
4) Choose Build Options from the Collection pop-up menu.
5) Choose DWARF for the Debug Information Format build setting.

You have two choices for DWARF support: DWARF, and DWARF with dSYM file. If you choose DWARF with dSYM file, Xcode creates a dSYM file that contains all the debugging symbols. Otherwise Xcode uses the symbols stored in the object files that were generated when Xcode compiled your program. Using a dSYM file is good for building the release version of your program. You can strip the debugging information out of the executable file to reduce its size. The debugging symbols are in the dSYM file in case you need them later.

There are some things to keep in mind if you decide to use a dSYM file. ZeroLink must be turned off. dSYM files cannot be used if you have a target that creates a static library or an object file. But you can use dSYM files for applications, frameworks, and dynamic libraries.

Debugging Command-Line Programs

For the most part debugging a command-line program is no different than debugging a GUI program. But command-line programs present one challenge. They require a console to enter input and display output. How do you enter input and display output when debugging a command-line program in Xcode?

The answer is to use Xcode's standard I/O log, which provides the console you need to enter input and display output. Choose Debug > Standard I/O Log to open the log window. You must start debugging your program before you can open the standard I/O log. Your program's output will appear in the log, and you will be able to enter any input your program needs.

Chapter 6

Shark

Windowed Time Facility

Shark 4.4 added the windowed time facility. Instead of storing every sample, profiles that use the windowed time facility store only the most recent samples. You get to specify how many samples to store in the recent sample history.

Why would you want to store only the most recent samples? Storing the most recent samples lets you profile for a long time. The Shark profiles that store every sample are designed to profile your program for a short time, no longer than a couple of minutes. If your program starts to run slowly after running for 20 minutes, you're out of luck because Shark stopped sampling before the slowdown occurred. By using the windowed time facility, you can keep your program running until your program starts to run slowly. When your program starts to run slowly, you can pause Shark, look at the recent samples, and see what caused the slowdown.

Setting Intel Performance Events

Shark 4.4 replaced the Pentium M option with the Intel Core and Intel Core 2 architecture options. 32-bit Intel Macs have the Intel Core architecture, and 64-bit Intel Macs have the Intel Core 2 architecture.

Chapter 11

Dealing with Binary and Bundled Files

I clarified what to do when checking the `cvswrappers` file back into the repository. You must navigate to the `CVSROOT` directory to check `cvswrappers` because `cvswrappers` resides in the `CVSROOT` directory.