1 Introducing Xcode 5

Xcode is Apple's free suite of developer tools; it is used to create applications for iOS mobile devices and for Mac OS X. Xcode 5, shown in Figure 1.1, is the most recent version.

Developer tools are complex, and Xcode has always tried to hide much of that complexity from novice developers. You can use Xcode in a very simple click-to-build way, but this simplicity can be misleading. Many developers never explore Xcode's more advanced features and never discover how they can use them to save time, solve problems, or extend their projects with original and creative features.

Xcode also includes an unexpectedly enormous selection of helper applications and developer tools. A complete guide to every element in Xcode would require a shelf of books. This book concentrates on beginner- and intermediate-level features, but it also includes hints and pointers for advanced developers.

COPYRIGHTED MATERIAL

Understanding the History of Mac Development Tools

1 In This Chapter

- Understanding the history of Mac development tools
- Moving to Xcode 5
- Comparing iOS and OS X development

Before OS X, Apple's IDE (Integrated Development Environment) was MPW (Macintosh Programmer's Workshop). MPW is no longer supported, but can still be found on the Internet Archive, as shown in Figure 1.2.

MPW competed with a commercial development system called CodeWarrior. Both were expensive, and many would-be developers were put off by the initial start-up costs.

Looking back at early IDEs

CodeWarrior was based on the Metrowerks C compiler and environment. It smoothed the transition from the 68k processors to the PowerPC and helped make the new PowerPC Macs a success. As an IDE, CodeWarrior provided complete support for the PowerPC architecture; MPW took longer to catch up with Apple's own new hardware. CodeWarrior also compiled code more quickly than MPW and created faster and more efficient binaries.

Figure 1.1

Xcode 5 is a significant refresh of the radical changes introduced in Xcode 4.

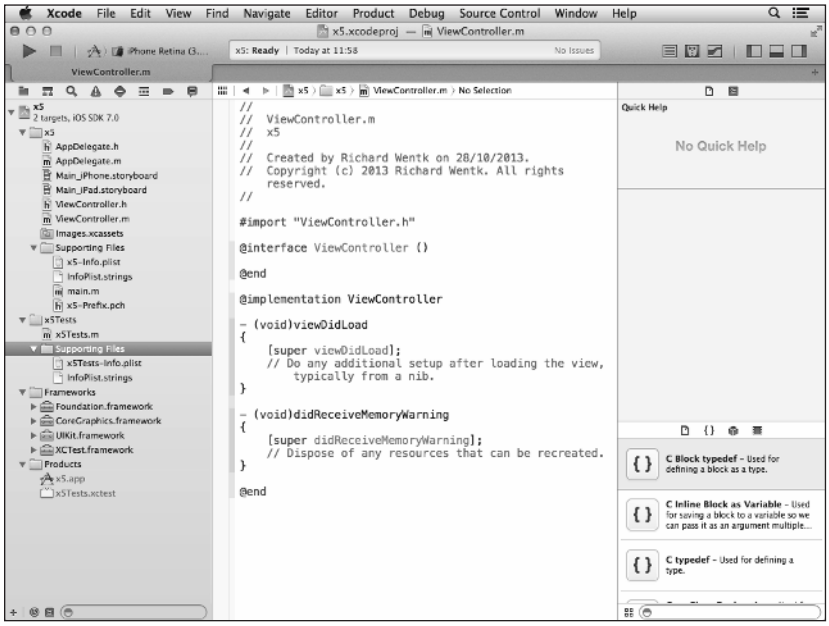
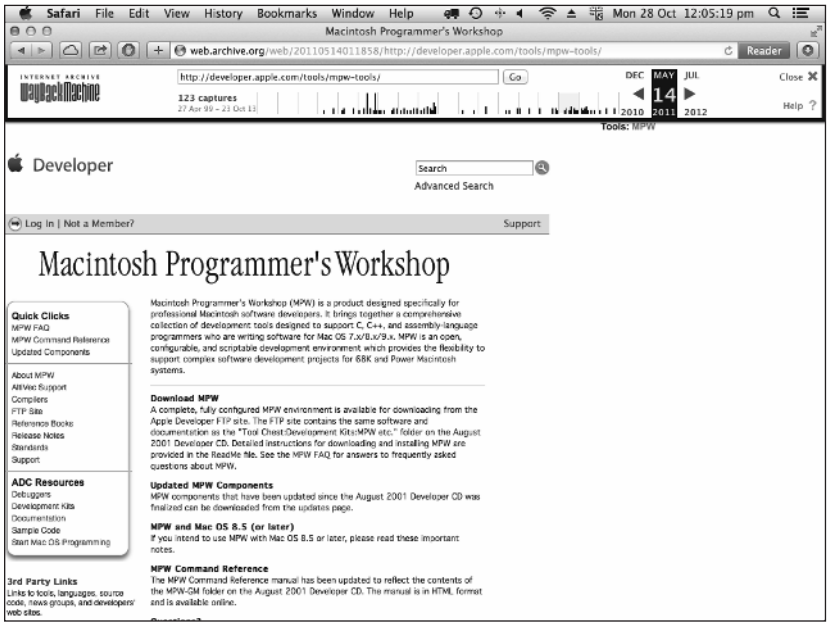


Figure 1.2

The MPW IDE is no longer available from Apple, but users of antique Macs can still download it from other sites.





NOTE

Early versions of MPW were famous for their error messages, which included "We already did this function," "This array has no size, and that's bad," and "Call me paranoid, but finding '/*' inside this comment makes me suspicious." Later Apple IDEs reverted to more traditional messages.

Developing Xcode

With the move to OS X, Apple decided to retain control over the developer environment. An IDE called Project Builder had been developed as part of the NeXTStep project. A free copy of Project Builder was bundled with every copy of OS X. In Fall 2003, an updated and enhanced version was shipped and named Xcode 1.0.

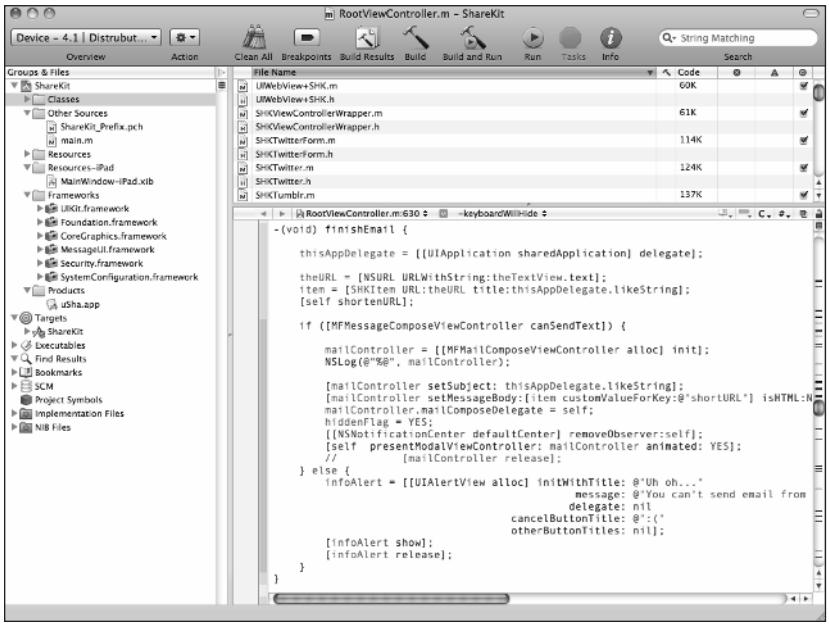
Xcode has been updated with every major new release of OS X. Xcode 2.0 shipped with OS X 10.4 "Tiger." It included improved documentation, better support for Java, and the Quartz Composer visual programming tool, which is described in more detail in Appendix B.

Xcode 3 shipped with OS X 10.5 "Leopard" and introduced improved debugging tools. Xcode 3.1 added support for early versions of iOS.

Xcode 3.2 is shown in Figure 1.3 and was released with OS X 10.6 "Snow Leopard." Prior to this release, Apple supplied separate builds of Xcode for iOS and OS X development. With version 3.2, Xcode became a unified development environment that could target both platforms. This widened the developer base, but it also made Xcode more difficult to download. The Mac version was around 800GB. The combined version is typically around 3GB.

Figure 1.3

The Xcode 3 IDE was productive but limited by obvious UI inefficiencies, such as poor support for editing multiple files simultaneously.



Alternatives to Xcode

Xcode is optimized for visual development of Objective-C and Cocoa projects. In practice, this means the Cocoa and Cocoa Touch libraries and documentation are tightly integrated into Xcode. Xcode 5 supports Objective-C++, which allows developers to mix C++, C, and Objective-C code, with some limitations. For example, the C++ and Objective-C class hierarchies are independent and do not allow cross-inheritance.

For details of mixing and interfacing Objective-C and C++, search online for Chapter 13 of "The Objective-C 2.0 Programming Language"—a document originally written and distributed by Apple, but now available only from other sites.

If you are used to developing in a different environment, you may feel that Xcode works in ways that don't match your requirements. If you plan to create windowed applications with official Apple UI elements, building Objective-C and Cocoa code in Xcode is likely to be your most efficient choice. If you prefer to create UNIX applications with command line or X11 features, you may prefer an alternative.

Although OS X is based on Darwin/POSIX rather than Linux, it's relatively easy to create a cross-platform application core that can be extended with platform-specific features. It's possible to use Xcode from the command line in Terminal with your own make files (build management and configuration files). If you're used to GCC and GDB on other platforms, you can run them directly from the command line, bypassing most of Xcode's features.

Java and C/C++ developers may prefer the free Eclipse IDE available at `www.eclipse.org`. Eclipse can be extended with a C/C++ IDE. Cocoa isn't supported, but Java and mixed development are.

For multi-platform support, Mono remains an option. Mono compiles C# rather than Objective-C or C++, but it's designed to support cross-platform output, running similar code on Windows, OS X, iPhone, Android, and Linux platforms. Mono also supports ASP.NET web projects.

MonoMac and MonoTouch versions include bindings to key OS X and iOS APIs. A version for Android is also available. The main IDE is called MonoDevelop and is available at `monodevelop.com`. Although Mono has obvious advantages, Apple's support for the competing platform isn't reliably enthusiastic. At times, Apple has barred from the App Store apps developed in languages other than C, Objective-C, and C++. But some MonoTouch applications have been approved for sale. Mono may be a better choice for developers coming from a C# and Windows background who don't want to learn a completely new language.

Game developers should explore the ever-increasing number of cross-platform game libraries and development environments that support iOS and OS X. Examples include Unity, Sparrow, Citrus, and cocos2D. The state of game development changes continuously. For the most recent details, search online for "iOS game engine" and "OS X game engine," adding other platforms—Android or others—as needed.



CAUTION

Strong Java support was a feature of earlier Xcode versions, but that has been downgraded in recent releases. Apple has moved Xcode toward supporting C-family development, including C, Objective-C, C++, and Objective-C++. These are now the officially supported languages for iOS and OS X development.

Understanding Xcode 5's Key Features

For developers who are beginning Xcode, Xcode 5 includes the following features:

- A project navigator that lists and groups related project files
- File and project templates for both OS X and iOS projects

- A code editor that includes static code checking, code completion, and dynamic hints and tips
- A visual UI design tool called *Interface Builder,* also known as IB, which can prototype visual interfaces, but can also be used to manage and preload other application objects
- Further integrated editors for class management and for Apple's Core Data database framework
- A debugger that supports expressions and conditional breakpoints
- Support for direct access to various online code repositories
- A minimal but useful *iPhone Simulator* that runs iOS applications on a Mac
- A collection of *Instruments*—tools that can profile speeds, monitor memory allocations, and report other key features of code as it runs
- Support for both visual development and low-level command-line compilation
- A selection of further helper applications



CROSS-REFERENCE

For a list of helper tools and applications, see Appendixes A and B.

Xcode doesn't support or include the following:

- **Editors for graphics, sounds, fonts, 3D objects, or other media types:** External editors must be used.
- **Built-in support for languages other than C, C++, and Objective-C:** You can extend Xcode to work with other languages, but Xcode is optimized for C-family development. (This does not include C#.)
- **Development tools for other operating systems:** OS X is similar enough to BSD UNIX to allow for some code sharing. But Xcode cannot be used to develop applications for Windows, Android, or Linux, or for web languages such as Perl and PHP.
- **Unlocked open development for iOS:** Applications for iOS hardware must be code signed and linked to a time-limited certificate. In practice, this means that even if you use Xcode, own an iPhone, and are a registered developer, your own applications will cease to run after the time-limited certificate expires.
- **Development on non-Apple platforms:** Currently, Xcode requires a Mac running a recent copy of OS X.



NOTE

Rumors come and go of a merger, or at least a relationship, between Xcode and Microsoft's Visual Studio development tools. At the time of this writing, a merger seems almost impossibly unlikely.

Code changes in Xcode 5

Xcode 5 includes some significant changes to Objective-C. Older code still compiles, but you can use some new time-savers when writing new code.

The first change is that ivars are now optional. If you declare properties in the usual way, you should not need to add corresponding ivars—the compiler generates them automatically.

You can also save time with a new autosynthesize feature. `@synthesize` directives are now optional. Xcode can add `@synthesize` directives automatically.

This new feature has a catch—you need to add an underscore in front of autosynthesized property names. For clarity and compatibility, you may want to continue using explicit directives.

Moving to Xcode 5

Xcode 5 simplifies and expands many of the features in Xcode 4. Many developer tasks are repetitive chores that have become embedded in the development process for historical reasons. Developer tools typically assume a workflow and mindset that date back to the very earliest days of computing, more than half a century ago.

The designers of Xcode 5 have continued to rethink these assumptions and tried to automate chores where possible. Compiler technology has also improved, and Xcode 5 no longer supports the GNU compiler and debugger; it works exclusively with a newer compiler toolchain called LLVM.

Compared to Xcode 4, Xcode 5 has a number of improvements:

- **Simplified and automated configuration:** Xcode 5 makes it easier to manage devices, teams, and development certificates. It also includes built-in tools for setting up apps for test- and runtime-access of Apple services such as iCloud, Passbook, and Game Center.
- **Enhanced testing:** A new Test Navigator supports simplified development and management of unit tests.
- **Continuous integration bots:** When used with OS X Server for OS X Mavericks, developers can create "bot"' that build apps, check and test them, and distribute them automatically to beta testers.
- **Improved layout tools:** The old nib file system has been integrated into screen-oriented storyboards, which simplify UI (User Interface) development. Each screen includes auto-layout features that make it easier to produce layouts that work with different screen resolutions, aspect ratios, and orientations.
- **Simplified asset management:** Icons, launch screens, and other art can be collected and compressed into a single Asset Catalog file.

- **Improved debugging:** Gauges show CPU, memory, and other resource requirements, including energy use. A new visual debugger based on the LLDB engine makes debugging information easier to read. A Quick Look preview tool can show images, web pages, documents, and even Bezier curves in memory.
- **Simplified source control:** Branch management is streamlined, and you can choose to host and manage repositories locally, on OS X Server for OS X Mavericks, or on an external Git server.



CAUTION

If you load a project made with an older version of Xcode into Xcode 5, you sometimes see an error message reporting a "Missing SDK." There is an easy fix for this. For details, see "Selecting the Base SDK" near the beginning of Chapter 13.



NOTE

Xcode 5 is backward compatible with Xcode 3 and Xcode 4. You can load Xcode 3.x and 4.x projects, and you should be able to build them, after making any necessary updates to the code to make it compatible with the newest SDKs.

Introducing the Xcode 5 editor

The features of the editor are described in more detail in Chapter 3, but this chapter includes a simple orientation for impatient developers who are new to Xcode.

Xcode 5 gathers every feature into a single window with multiple work areas and panes. The active areas can be shown, hidden, split, or resized as needed. Every Xcode feature, including Interface Builder (IB) and the debugger, can appear in this window. Features can be hidden when you're not using them. You can also create your own *workspaces* to save and restore complete window layouts.



TIP

Xcode 5 is more productive with a larger monitor. You can have a console/debugger, editor, file list, and IB open simultaneously in tiled panes. With a large monitor, these panes become large enough to be truly useful without scrolling, resizing, or switching.

At the top of window, the toolbar area includes a summary panel that displays project status. This gives progress updates as a project builds and displays a count of warnings and errors after each build. The toolbar has been simplified. Only build/run and stop buttons are available.

Working with tabs

Xcode 5 uses *tabs*—editor sub-windows that work like the tabs in a browser, allowing single-click switching between selected files, as shown in Figure 1.4. With tabs, you can add your choice of files to the tab bar as you work and then save the tab bar with the project. You can also remove files from the tab bar when you are no longer working on them.

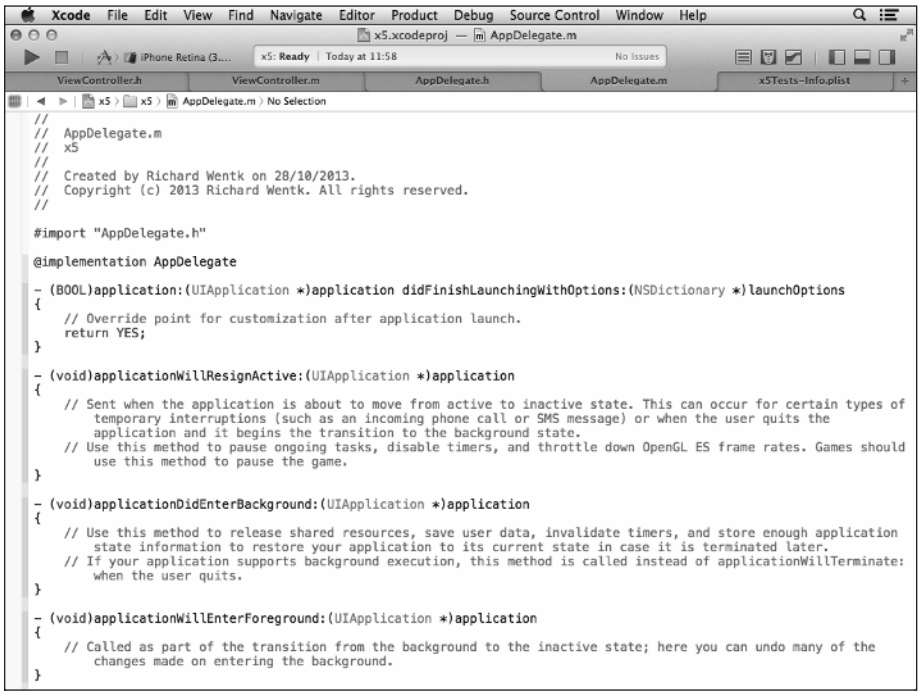


TIP

In Xcode 5, you can show/hide the Navigator area at the left of the window and the Utilities window at the right independently for each tab.

Figure 1.4

The tab bar appears under the main toolbar near the top of the Xcode 5 window.





TIP

Tabs save the current cursor position, so you can use them to switch quickly between different sections of the same file. It's often useful to open multiple tabs that show the most significant methods or functions in a file.

Working with multiple windows

Not every developer is enthusiastic about single-window development. Fortunately, you can open multiple windows into a single project and select a different collection of editors and features in each window. A key goal is flexibility; you can arrange your workspace how you want it, with the features you want to see. As shown in Figure 1.5, you can still "tear off" a separate floating window to edit a single file by double-clicking the file.

Selecting and navigating files

Xcode 5 includes a hierarchical *navigation bar* that generates a menu tree from your project files, listing the files and symbols. As shown in Figure 1.6, you can select any file almost instantly. In fact, you can select any symbol in any file, for quick access to definitions and documentation.

Figure 1.5

In Xcode 5, you can still float individual files from a project in separate windows. But there are usually more efficient ways to work.

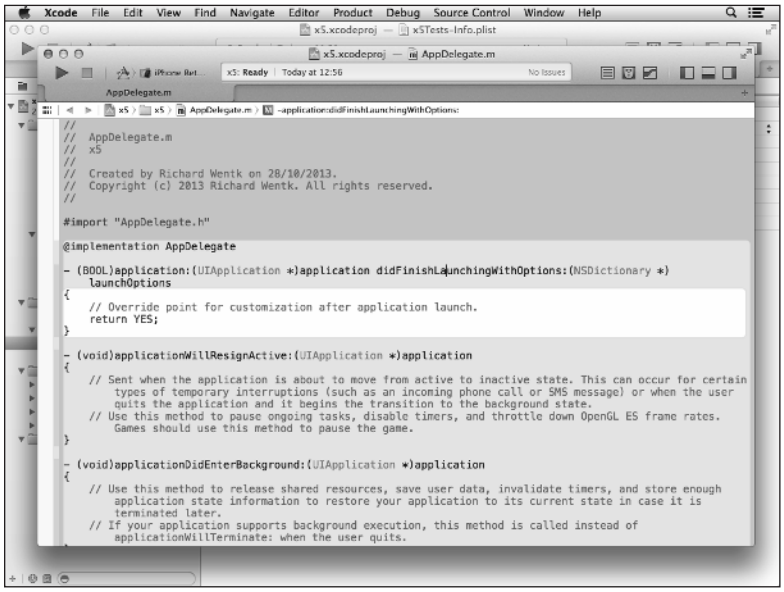
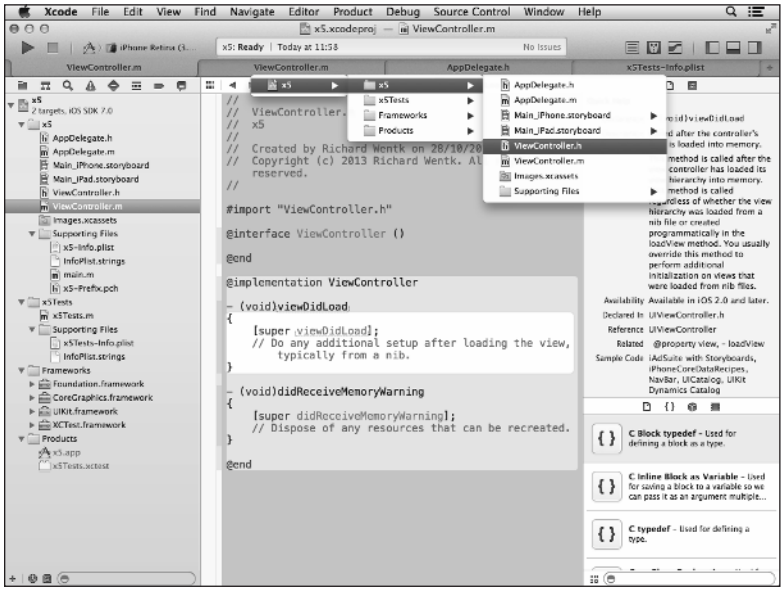


Figure 1.6

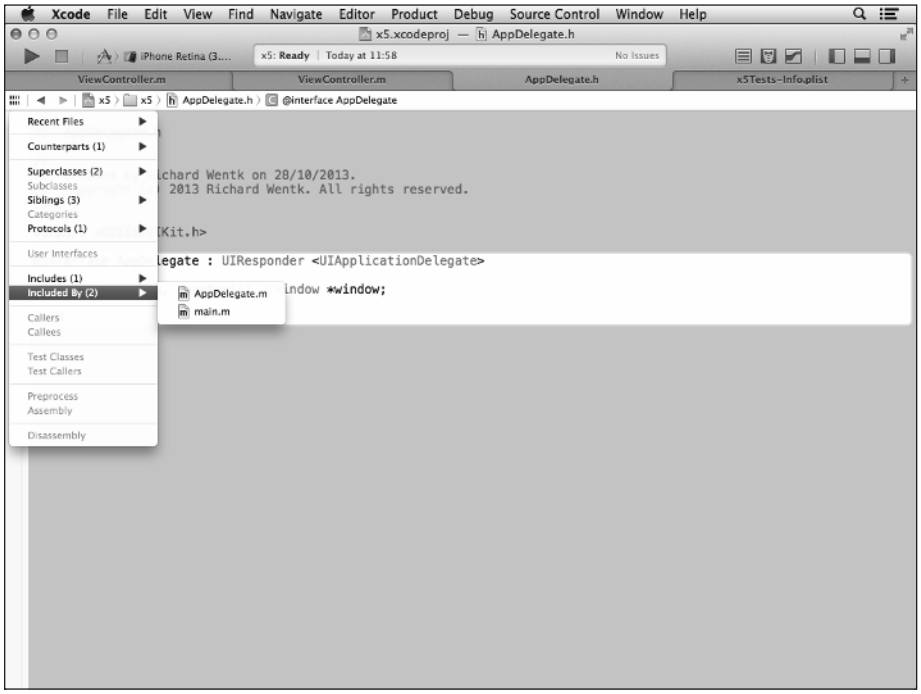
The navigation bar (*jump bar*) drastically speeds up access to any file in your project, by presenting them all in a single unified menu tree.



You can also select files in a more traditional Finder-like way using Xcode 5's Project Navigator. But as shown in Figure 1.7, you can access files in yet another way, through a separate menu that lists other relevant items, including header files, includes, related classes, and categories. Click the icon to the left of the left-pointing arrow to view this menu. It lists related files, including superclasses and subclasses, protocols, and includes.

Figure 1.7

At the left of the navigation bar, a separate menu shows files and items that are more loosely related to the currently selected file.

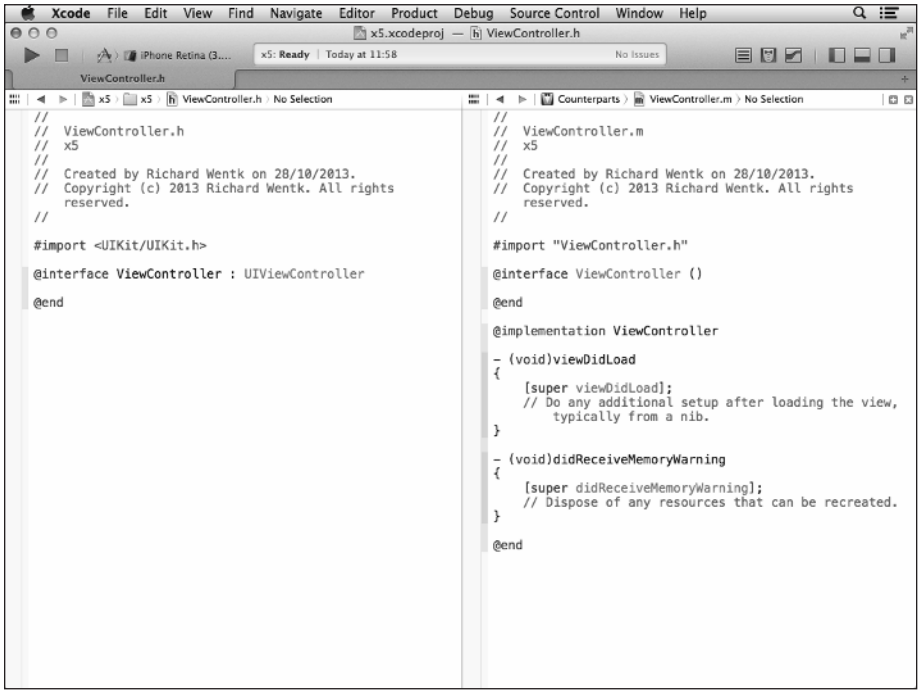


Using Counterparts

Xcode 5 uses an *Assistant View* to show related files in split windows. When you select a file for editing, Assistant makes an informed guess about a useful counterpart and displays it automatically. By default, this means that selecting a header file displays the implementation file in the second pane, and vice versa, as shown in Figure 1.8.

Figure 1.8

The button for selecting the Assistant option is in the grouping at the top right and looks like a waiter's dinner jacket with a bow tie. It splits the editor into two panes and automatically displays a counterpart file whenever a file is selected.





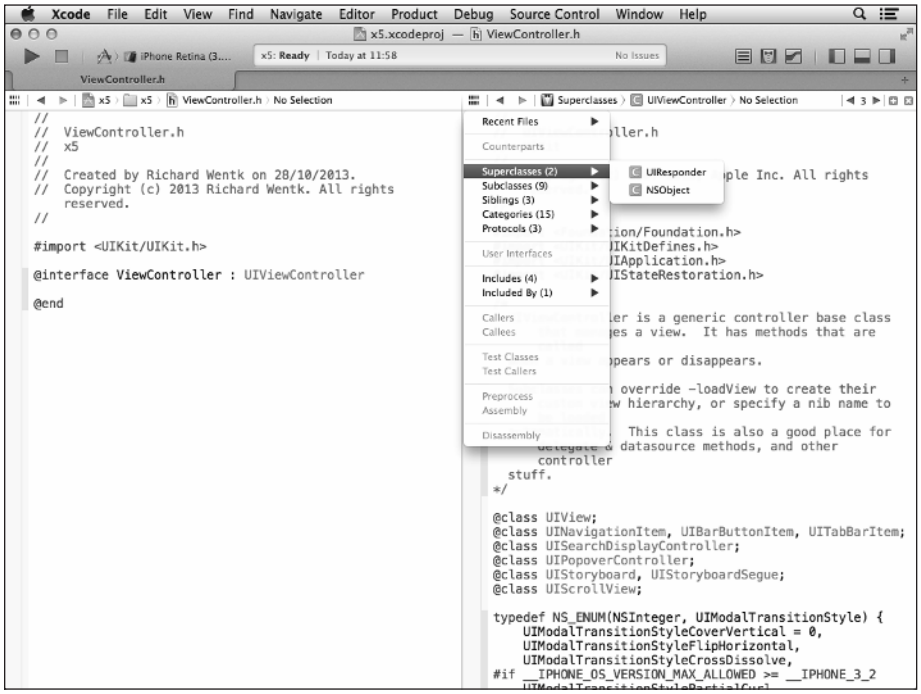
TIP

In Xcode 5, you can customize the behavior of the Assistant. You can split the pane as in Xcode 4, open a new editor pane, open a new window, or open a new tab for the counterpart. See the Navigation tab in Xcode ➪ Preferences. You can also set up double-clicks so they open a new tab or a new pane.

This feature is one of the most useful timesavers in Xcode 5. You can also manually select a counterpart or other file using a new contextual right-click menu, as shown in Figure 1.9.

Figure 1.9

You can change the behavior of Assistant to select a specific type of counterpart file, which can include an object's superclass as well as its headers and includes. This is useful for newcomers who may not be aware that Cocoa and other OS X headers are available in Xcode and can be used as a reference.



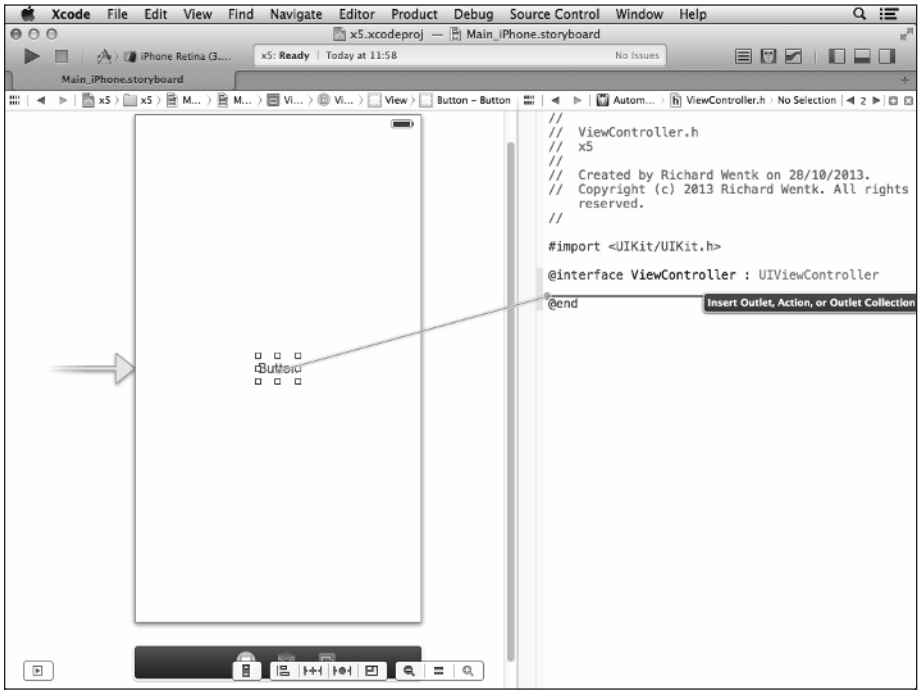
Working with Interface Builder

Interface Builder, Apple's UI development tool, is built into Xcode 5, and the UI layout process is closely integrated with associated code. One of the key Xcode development skills is learning how to link UI objects to the code that manages their behavior. In Xcode, you do this by control-dragging a line from a UI object to a view controller and telling Xcode to insert some skeleton definitions for the associated code, as shown in Figure 1.10.

Xcode inserts appropriate code for you in both the header file and the implementation. It also synthesizes outlet variables automatically. For detailed examples of creating links between outlets, actions, and IB objects, see Chapters 7 and 8.

Figure 1.10

This powerful timesaving feature inserts outlet code automatically in Xcode 5.





CAUTION

In Xcode 5, iOS projects use storyboards for UI design, and OS X projects use the older nib file format. Although the technologies are different, the linking process used for both is very similar.

Exploring code and file management

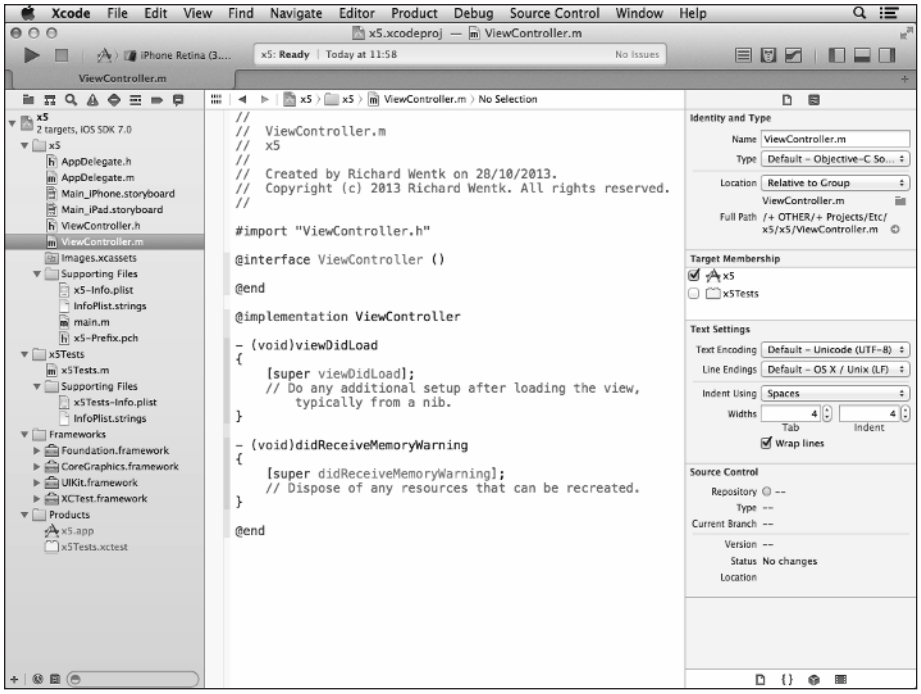
Xcode 5 includes two panes at the left and the right that can be revealed or hidden as needed, using a pair of buttons near the top right of the toolbar. The left pane, known as the *navigation pane*, lists various elements in a project. This includes a list of groups and files, as shown in Figure 1.11. But the pane can also list symbols, search tools, log listings, build issues, breakpoints, debugging information, and unit tests.

The pane on the right is the *utilities pane*. This pane gathers miscellaneous information; for example, IB's inspector panes appear here. It also shows build target and localization information, links to documentation, and file locations. When you select a file, the contents of this pane are updated automatically.

The utilities pane includes an optional Library sub-pane that can display file templates, standard code snippets, standard system objects that include both UI and data classes, and project media files. You can hide the Library sub-pane by dragging it down or reveal it by pulling it up.

Figure 1.11

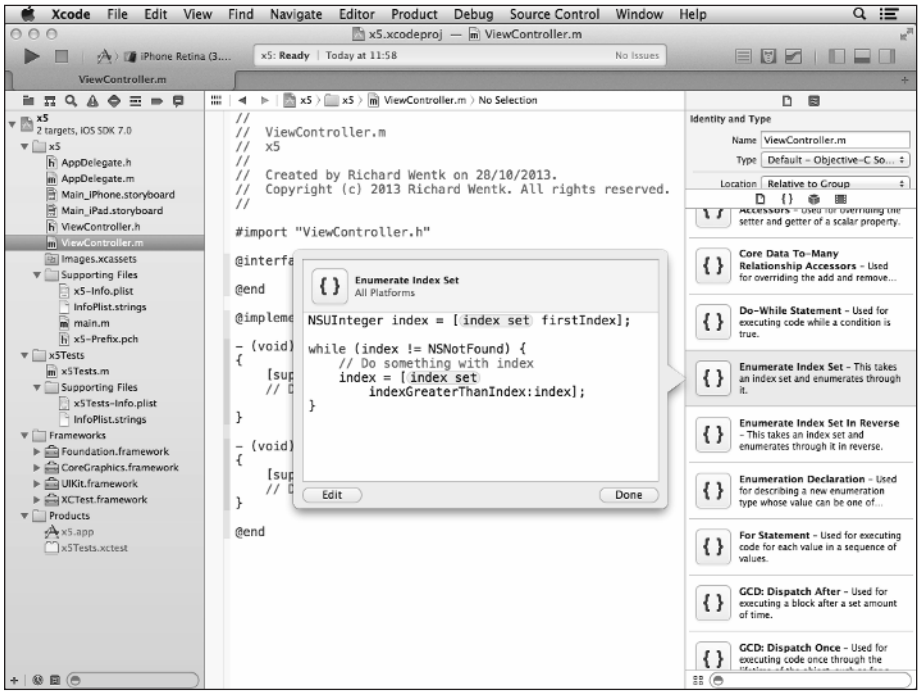
The left and right panes in Xcode 5 display ancillary information and manage optional features that may not be needed while editing.



The Code Snippet feature in the Library is shown in Figure 1.12. It's often useful to reuse the same code between projects, and the Code Snippet makes it easy to do this. To add code to a project, drag it from the library and drop it in the editor window. You also can view a preview of each snippet by selecting it before dragging. By default, this pane includes a small selection of standard snippets, but you can extend it indefinitely by adding your own. For more details, see Chapter 9.

Figure 1.12

The Code Snippets feature makes it easy to reuse code. You can add custom snippets of your own.

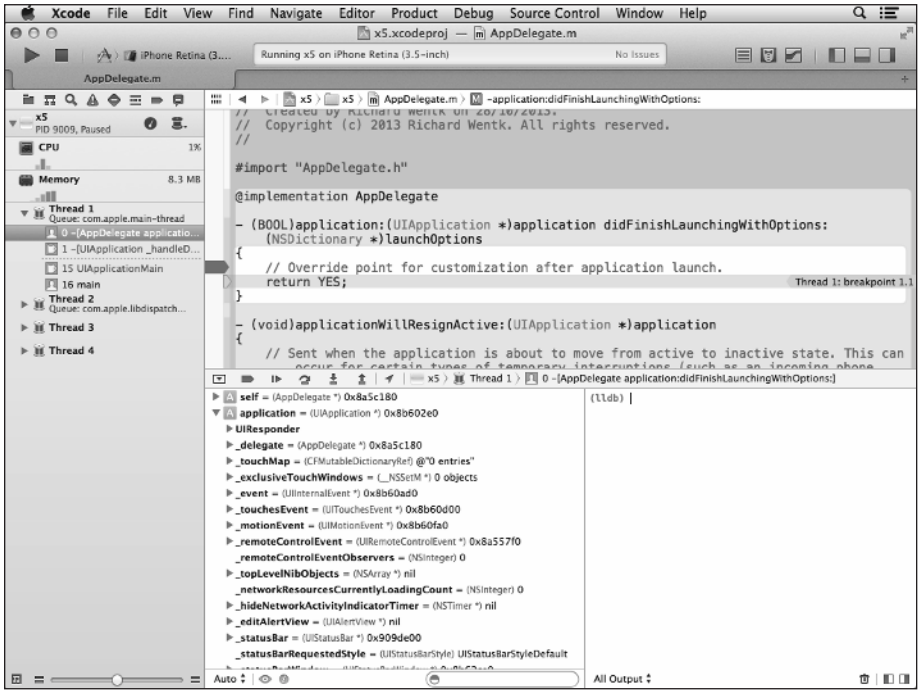


Exploring the debugger

As shown in Figure 1.13, the debugger appears in a new pane at the bottom of the editor window. To reveal it, select View➪Show Debugger Area. Both console output and debugger output appear in this area. You can choose to view either or both by clicking the new buttons that appear at the top right of the area. The debugger now supports multi-threaded debugging. You can set breakpoints by clicking in the gutter area to the left of the editor.

Figure 1.13

The debugger area appears in a pane in the main window, although for convenience you may decide to launch it in a separate window. On a smaller monitor, the debugging and console area can feel cramped.



Comparing iOS and OS X Development

Although Xcode supports OS X and iOS development equally and it can be used to develop apps for both the iOS and Mac App Stores, there are significant differences between the two platforms.

Developing for OS X

OS X development in Xcode 5 can be build-and-go, or you can choose to develop apps for the OS X App Store. Build-and-go development is unrestricted. You can create applications that run in a debugging environment on your own Mac and package them as applications that you can run independently, sell from a website, or prepare for network distribution.

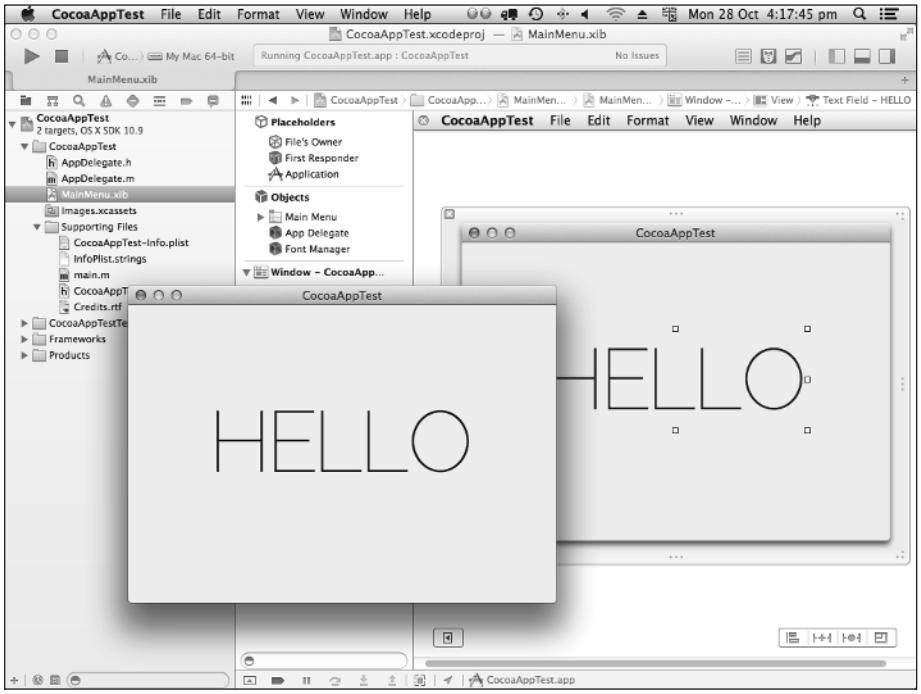
Optionally, you can *sign* them, so that OS X recognizes you as a registered developer when users try to install them. This is not obligatory, but it eliminates a warning message that may confuse users and discourage them from installing your application.

You can also create Mac apps for the App Store, which has its own separate distribution system. App Store apps must be signed and are installed automatically after purchase.

Figure 1.14 shows a simple OS X application using a template as a starting point.

Figure 1.14

You can create a very simple OS X application using a template as a starting point and adding a text label in IB. The application runs in its own window and replaces the OS X menu bar (not shown here). Although it appears to run independently, it is in fact controlled by Xcode and can be debugged while it's running.





NOTE

Xcode doesn't include network deployment features, but it does create application binaries that can be handed to network deployment tools.



CAUTION

Xcode 5 currently supports development of 64-bit applications for OS X 10.8 and 10.9. If you want to build 32-bit applications for OS X 10.6 or create universal binaries for older versions of OS X, you need an early version of Xcode 4.

Developing for iOS

iOS development is more complicated than OS X development. Development is controlled by *provisioning,* an Apple-generated security control, which is built into Xcode and manages access to hardware testing and App Store distribution.

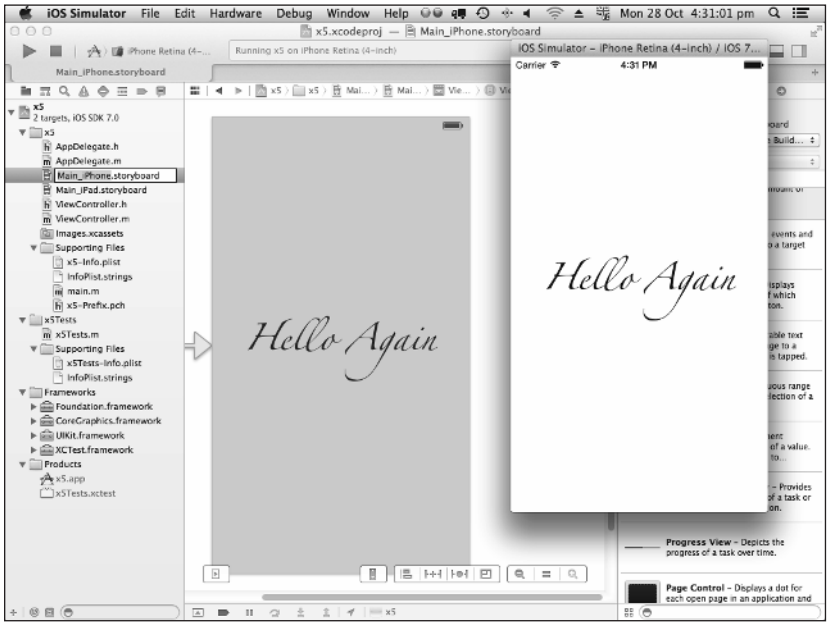
iPhone, iPod touch, and iPad platforms all use iOS, but these platforms are significantly different and may not always run the same version of iOS. Even when they do run the same version, not all hardware features and UI options are available on every device, and each device family now includes a wide range of display resolutions.

In extreme cases, conditional code is required to check which device an app is running on and which version of iOS it supports. Code paths may need to be selected accordingly with manual checks at runtime.

The Xcode Simulator, shown in Figure 1.15, includes separate iPhone and iPad testing options, with a selection of different resolutions and runtime environments. You can emulate GPS locations and journeys to a limited extent, but realistic emulation of hardware features such as the accelerometer and gyro is not available. All but the very simplest commercial apps should be tested on real hardware.

Figure 1.15

You can create a very simple iOS app and test it in the Simulator. The Simulator is best considered an educational rather than a production environment. It's adequate for apps with simple text and graphics, but it doesn't fully implement the GPS, accelerometer, gyroscope, or other hardware options in iOS devices.





NOTE

The Simulator in Xcode 3 and 4 included attractive graphic versions of iPhone and iPad hardware. Xcode 5 does not continue this tradition. The Simulator displays graphics in a plain window.

The extra requirements of provisioning and multi-platform support can make iOS development feel challenging and complex. A further complication is the need to produce high-quality supporting graphics and screen designs for maximum buyer impact in the App Store.

To date, Apple's beta cycle for iOS has been more aggressive than for OS X, and it's usual to have a new beta preview of iOS available almost as soon as the most recent version has been made public. Beta development requires a parallel version of Xcode and beta firmware for every test device. It may also require an updated version of iTunes and OS X.

So although the iPhone and iPad are simpler than a Mac, and app code can be much simpler, you should allow extra time to work for projects to support all the different possible targets, versions, and security options. You should also allow further time to develop attractive UI designs that work on all available screen resolutions and orientations.

Xcode 5 does not automate these options. The new layout tools can produce working multi-resolution UI designs, but the design process remains complicated. Developing and testing a universal iOS app—a single app that can run on an iPhone, an iPod, and an iPad, with support for all screen resolutions—remains a challenge.

OS X and iOS cross-development

In theory, you can migrate projects between platforms. In practice, Cocoa on OS X and Cocoa Touch on iOS have so many differences that dual-platform development of a non-trivial application is either challenging or unrealistic.

Currently iOS and OS X applications have distinct markets and partially distinct distribution models. Although a few applications have appeared on both—social networking tools and games are the most popular choices—Xcode does little to simplify the development of multi-platform projects.

The development workflow is almost completely distinct. The two platforms have these features:

- **A separate collection of classes for UI design and for data management:** Many of the more creative and sophisticated classes in OS X are either absent or only partially implemented in iOS.
- **A separate testing and debugging environment:** iOS applications can run in a Simulator or on a hardware device. OS X applications run in a debugging sandbox.
- **A different but converging core programming model:** iOS supports multi-tasking with some limitations and restricts access to the device file system. OS X supports a wider range of multi-tasking options and offers less restricted access to files, but you can build in restrictions if you choose to.

- **Separate documentation suites:** iOS and OS X have separate collections of documentation and distinct source code examples.
- **Different accelerated graphics frameworks:** OS X implements OpenGL in full; iOS implements the simpler OpenGL ES framework.
- **Different project templates for separate bare-bones starter application sketches:** iOS includes a set of templates for simple UI-driven handset apps. OS X includes a more complex collection of templates that support the development of plug-ins, screen savers, and other libraries.
- **A partially distinct set of supported instruments for testing:** Limited overlap exists, but some instruments remain unique to each platform.

Apple's Model-View-Controller design pattern implies that applications should keep UI designs, UI management code, and underlying data collections distinct. OS X includes controller classes that make it easier to manage data and create UIs to work with it. Most of these classes are absent in iOS. This makes it difficult to use MVC effectively when attempting dual-platform development.

If you plan to develop across platforms, try to package the underlying data model into its own collection of classes. You may be able to reuse these classes without major changes. Keep UI and UI management code elsewhere.

Generally, combined iOS and OS X remains possible but difficult. It's more realistic to think of Xcode 5 as two separate development environments with a common frontend than as a single unified environment designed to produce code for either platform.

Summary

Xcode 4 was a dramatic reinvention of Xcode 3, but Xcode 5 is more of a slipstream update with fixes, consolidations, and minor changes. It's cleaner and simpler than Xcode 4, and it should be easier to work with. But a few key features have been moved or even removed. Expect a short period of readjustment, but no significant shocks.