1 Introducing Xcode 4

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 4, shown in Figure 1.1, is the most recent version and is a radical update with many new features.

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.

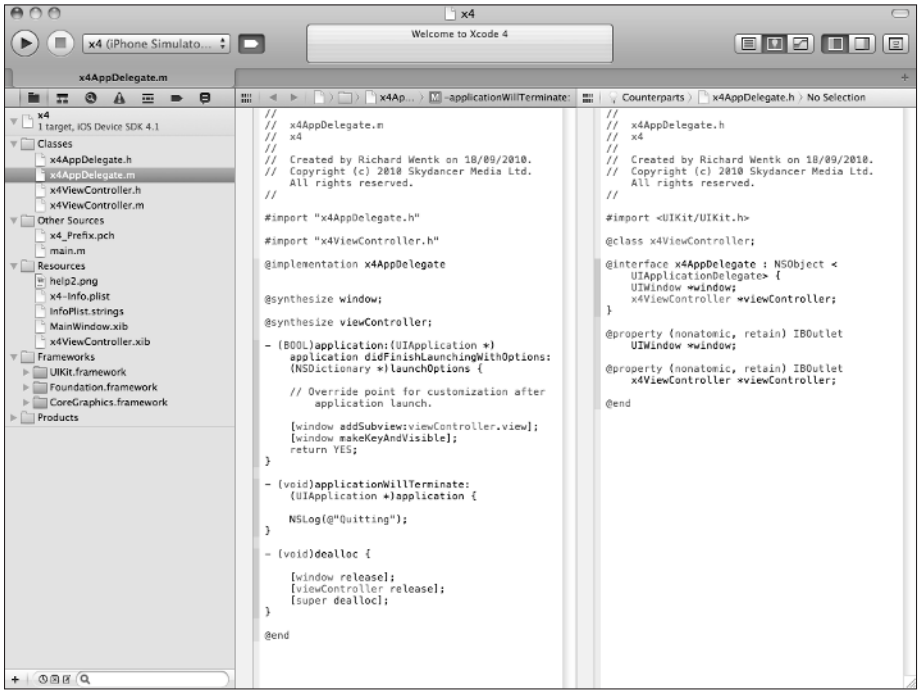
Xcode also includes an unexpectedly enormous selection of helper applications and developer tools. A complete guide to every element in Xcode would require a shelf of books. This book concentrates on beginner- and intermediate-level features, but also includes hints and pointers for advanced developers.

1 In This Chapter

- Understanding the history of Mac development tools
- Developing Xcode
- Moving to Xcode 4
- Comparing iOS and OS X development

Figure 1.1

Xcode 4's simplified interface hides familiar old features and some unexpected new ones.



Understanding the History of Mac Development Tools

Before OS X, Apple's IDE (Integrated Development Environment) was MPW (Macintosh Programmer's Workshop). As shown in Figure 1.2, MPW, which is still available today, was in competition with a commercial product called CodeWarrior. Both CodeWarrior and MPW 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 1.2

The MPW IDE is available on Apple's FTP site, and users of antique Macs can download and use it.

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.

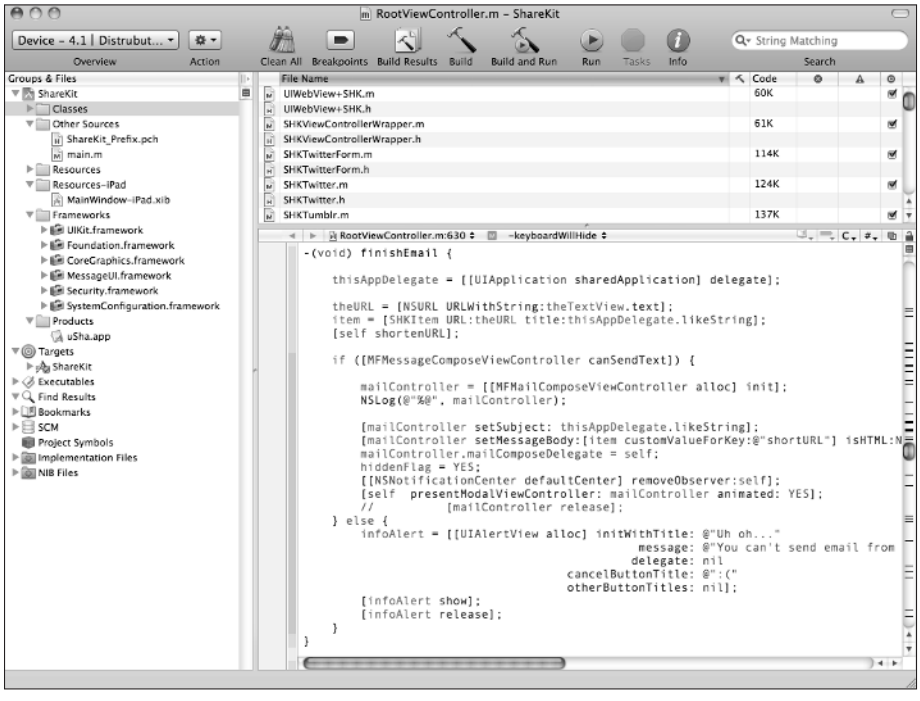
Xcode has been updated with every major new release of OS X. Xcode 2.0 shipped with OS X 10.4 "Tiger." It included improved documentation, better support for Java, and the Quartz Composer visual programming tool, which is described in more detail in Appendix A.

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 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.



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.

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 4 has moved toward improving support for C++, but there are still limits to how easily it's possible to mix Objective-C, Objective-C++, and C++ code, Apple's own libraries and example source code are a combination of traditional C and Objective-C. C++ and Objective-C++ aren't widely used.

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 a popular choice. Mono compiles C# rather than Objective-C or C++, but is 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.

On the iPhone, Flash developers can package Flash projects as iPhone applications with Adobe's Packager for iPhone. Originally included in various versions of Adobe CS5, Packager was withdrawn when Apple restricted iPhone applications to native Objective-C and C++ code. Apple subsequently lifted the restrictions later in 2010 and at the time of writing Packager is available as a free beta project from the Adobe Labs site at `labs.adobe.com`. Future production versions are likely to have their own URL and product pages.

iPhone game developers may also want to consider Ansca's Corona, which is a simplified scripted development environment for iOS and Android available from `anscamobile.com/corona`. Corona currently costs $349 per year, but claims faster development times than are possible with Xcode and native Objective-C.

Understanding Xcode 4's Key Features

For developers who are beginning Xcode, Xcode 4 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.
- An impressive selection of further helper applications that aren't built into the main Xcode interface but are installed with Xcode and can be run independently, as needed. The tools include a packager for building installable OS X applications, hardware monitoring and testing, an animation design tool, a tool for building JavaScript widgets that can be distributed commercially, and others.

CROSS-REFERENCE

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

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 surface regularly of a merger, or at least a relationship, between Xcode and Microsoft's Visual Studio series of development tools. There would be obvious commercial benefits to allowing Windows developers access to iOS and the App Store, but Apple's culture tends to be closed and proprietary. A formal link is possible, but at the time of writing it seems very unlikely.

Moving to Xcode 4

Xcode 4 marks a significant change, because the aim is no longer to produce code, but to simplify the developer experience. 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 4 have begun to rethink some of these assumptions, adding features that can streamline and simplify the workflow. Some of these features are specific to Cocoa and Objective-C development, while others are more general improvements in code management and debugging. Compiler technology has also improved, and Xcode 4 is moving toward the latest and fastest compiler tools.

CROSS-REFERENCE

For details of the compiler technologies available in Xcode 4, see Appendix C.

Compared to Xcode 3, Xcode 4 has a completely redesigned interface and a selection of extra features:

- A unified interface in a single window
- Integrated editors for all main code and data file types
- Integration of *Interface Builder,* the Xcode visual UI design tool
- Simpler and faster navigation between files
- Integrated code management with version control, repository access, and a code library to simplify reuse of code snippets
- Improved debugging and code testing with more informative error messages, static code testing, support for multiple log files, and better Code Completion (formerly Code Sense) hinting
- New code analysis features, which offer hints about basic coding errors
- Improved and simplified code and symbol searches
- Simplified management of build targets and products

- Support for *Schemes,* which offer fine control over how projects build, and support different build options for different applications (for example, testing, debugging, packaging for distribution, and so on)
- Support for *Workspaces,* which make it easy to manage and work on multiple related projects
- Backward compatibility with Xcode 3.*x* project files
- Improved compiler technology
- Various other time-saving features, such as automatic unprompted file saves before a build

CAUTION

If you load a project made with an older version of Xcode 3 into Xcode 4, you sometimes see an error message reporting a "Missing SDK." You can't build the project until you fix this. For details, see "Selecting the Base SDK" near the beginning of Chapter 13.

Moving from the Xcode 3 to the Xcode 4 editor

The new features of the editor are described in more detail in Chapter 3, but this chapter includes a simple orientation for impatient developers who are already familiar with Xcode 3. In outline, almost all the familiar features have been retained, and there are some new arrivals. But the UI has been reorganized, and features may be in unfamiliar locations.

NOTE

Xcode 4 is backward compatible with Xcode 3. You can load Xcode 3.*x* projects and save them again, and Xcode 3 should still be able to open them. Naturally, you can't open Xcode 4 projects in Xcode 3.

In Xcode 3, floating windows could proliferate uncontrollably, making them difficult to work with. Xcode 4 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. Hiding and revealing features adds a small overhead, but is much more efficient and productive than a chaotic mess of windows. You can also create your own *workspaces* to save and restore complete window layouts.

TIP

Compared to Xcode 3, Xcode 4 becomes more efficient with a larger monitor. Xcode 3 often wasted screen real estate; for example, the right side of a code window was usually empty. With Xcode 4, 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 new 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.

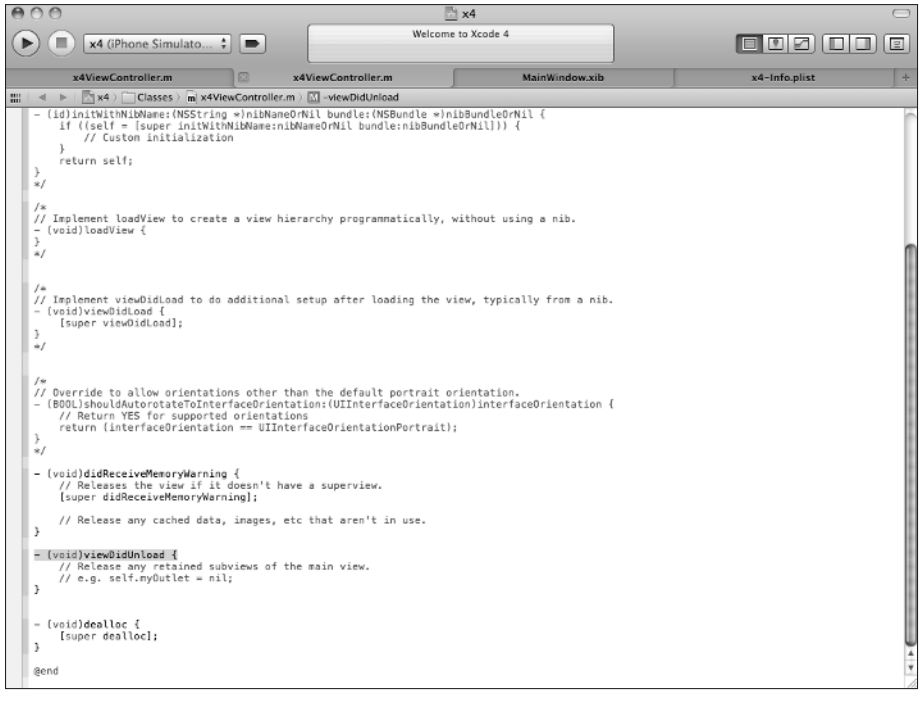
In the first release of Xcode 4, it's no longer possible to customize this area with your own selection of build/run/stop/clean options, as it was in Xcode 3.

Working with tabs

Xcode 4 introduces *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. Tabs replace the file list that appeared above the editor pane in Xcode 3. The file list was an inefficient way to select files for editing. 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 delete files from the tab bar when you are no longer working on them.

Figure 1.4

The new tab bar replaces the project file list and appears under the main toolbar near the top of the Xcode 4 window.



TIP

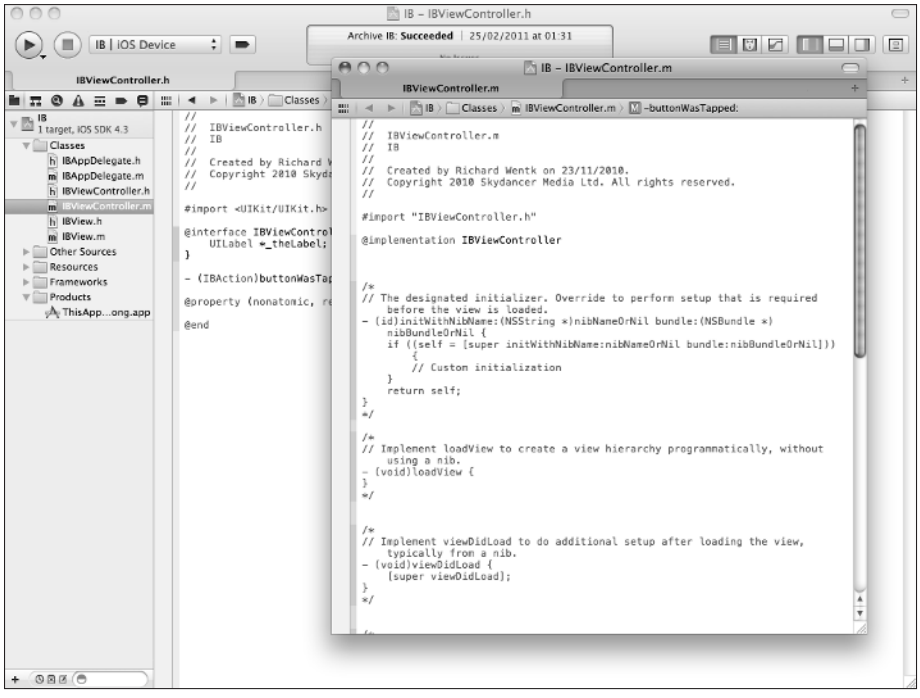
Tabs save the current cursor position, so you can use them to switch quickly between different sections of the same file. It's often useful to open multiple tabs that show the most significant methods or functions in a file.

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 create a separate floating window to edit a single file by double-clicking the file.

Figure 1.5

In Xcode 4, you can still float individual files from a project in separate windows. But there are usually more efficient ways to work.

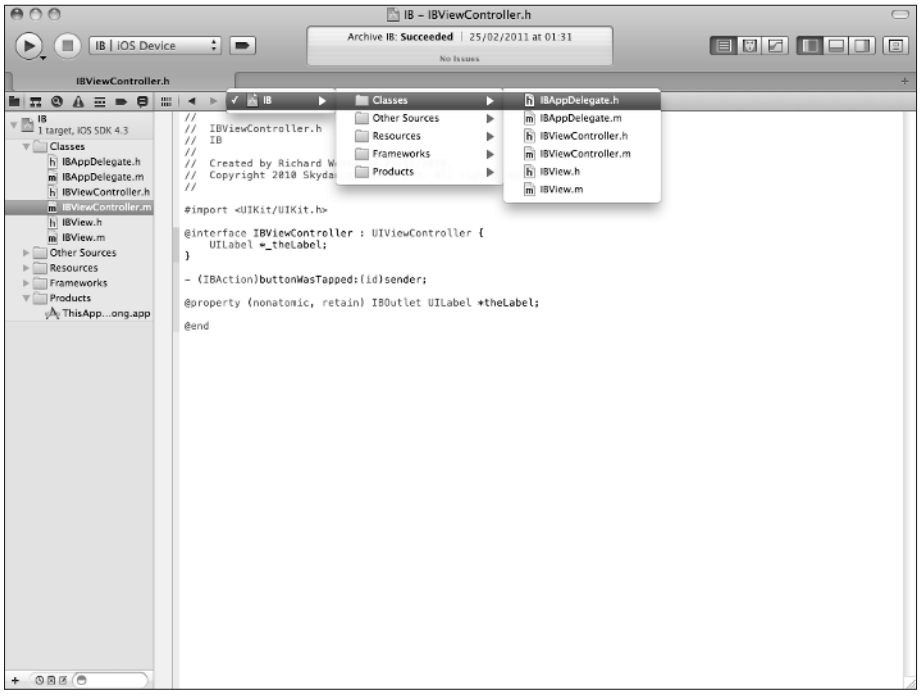


Selecting and navigating files

Xcode 3 included a pair of file and symbol navigation menus above the main editing pane. Xcode 4 extends this idea and displays 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.

Figure 1.6

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



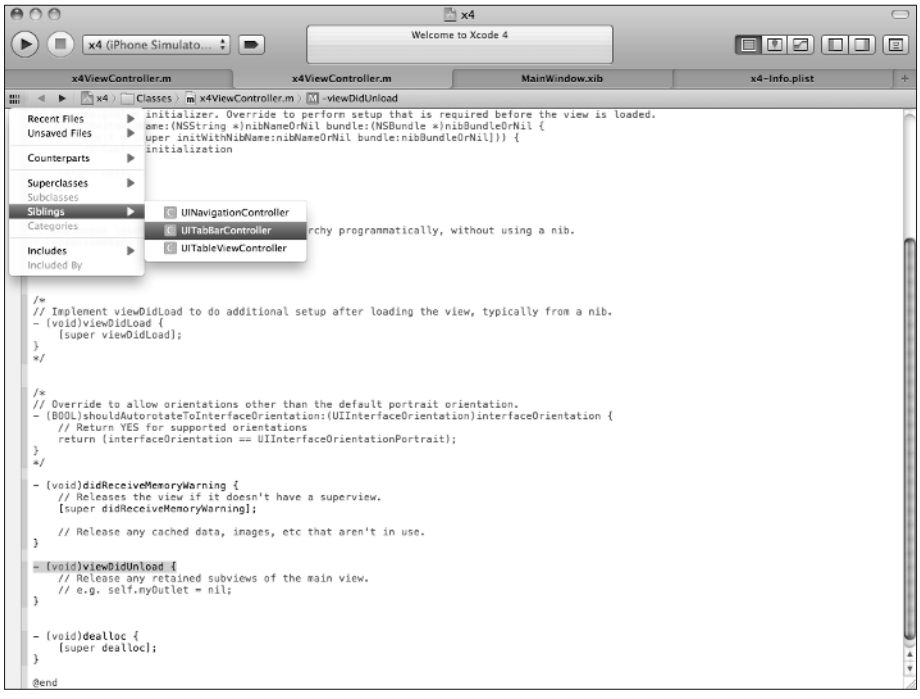
NOTE

In beta versions of Xcode 4 you could use the menu tree to navigate to the methods in each file. This feature was removed in the final release. It was a *very* useful feature, so it may return in future updates. Symbols still appear in the navigation bar in a separate menu, much as they did in Xcode 3.

You also can select files in the traditional way using Xcode 4's version of the Groups & Files pane, which is now called the Project navigator. But the navigation bar is very much faster. As shown in Figure 1.7, it includes a separate menu that lists other relevant items including header files, includes, related classes, and categories. Click the boxes icon to the left of the left-pointing arrow to view this menu.

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.

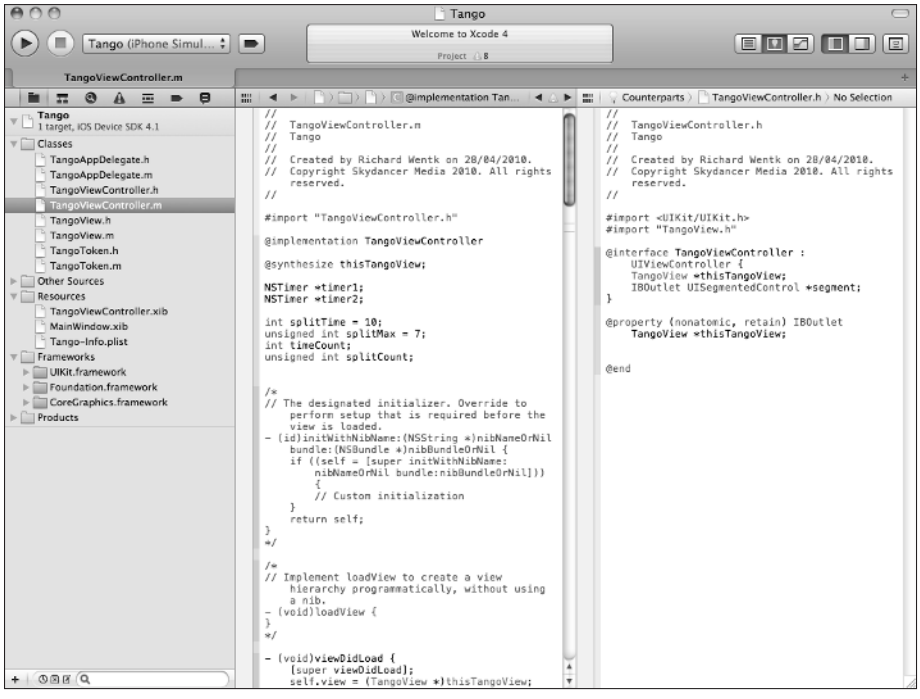


Using the Assistant

Xcode 3 included a *counterpart file* selector that switched an editor window between a class header and its corresponding implementation file. Xcode 4 introduces *Assistant*, which is designed to work with a new vertically split double-pane view. When you select a file for editing, Assistant makes an informed guess about a useful counterpart and displays it automatically, as shown in Figure 1.8. By default, this means that selecting a header displays the implementation file in the second pane, and vice versa.

Figure 1.8

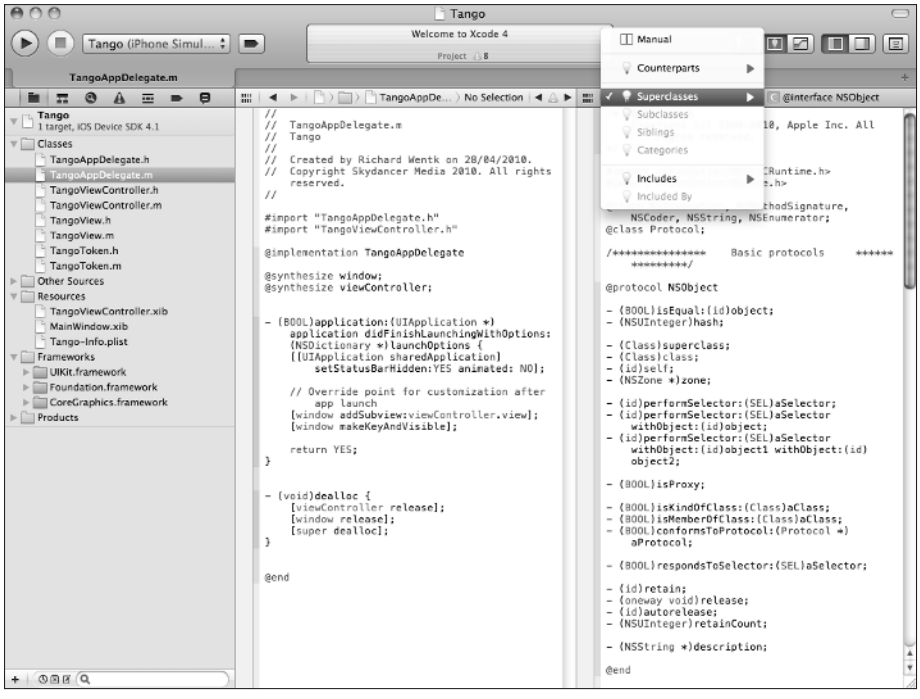
The button for selecting the Assistant option is in the grouping at the top right and looks like a light bulb. It splits the editor into two panes and automatically displays a counterpart file whenever a file is selected.



With the vertical split view and Assistant, you no longer need to manually switch between counterparts or to work with the less efficient horizontal split view available in Xcode 3. This feature is one of the most useful timesavers in Xcode 4. You also can manually a 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.



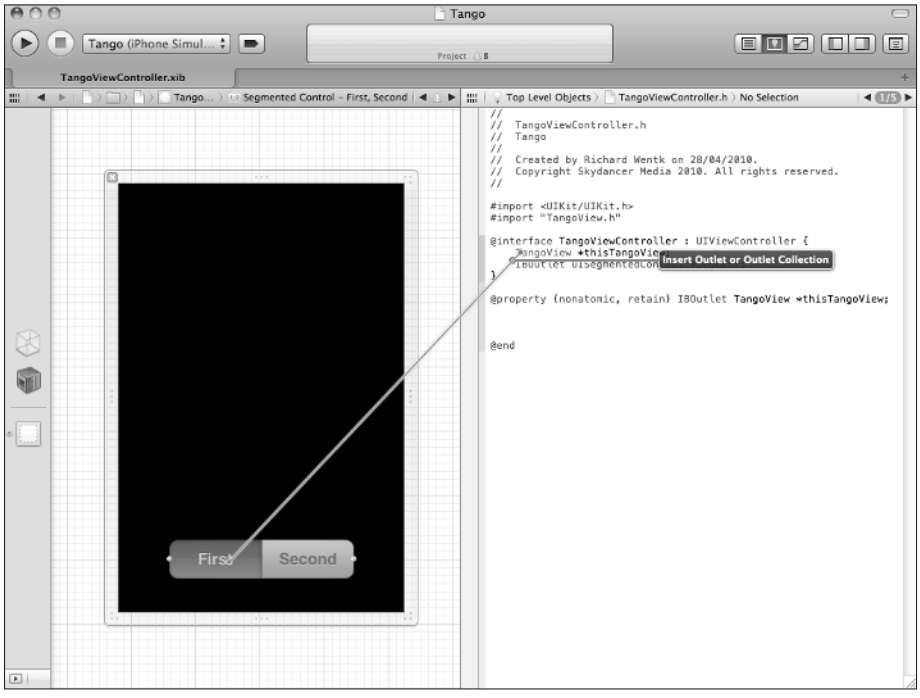
Working with Interface Builder

IB is now built into Xcode 4. It launches at the same time as Xcode, and you can edit a nib file simply by selecting it. IB in Xcode 3 had the same poor window management as the main editor. IB's windows would often hide behind other windows for no reason.

In Xcode 4, you can use tabs, the navigation bar, and other new editor features to work with multiple files more efficiently. Linking and symbol editing have also been drastically simplified. You can drag links directly from a control or object in IB to a code window, 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 among outlets, actions, and IB objects, see Chapters 7 and 8.

Figure 1.10

Creating outlet code automatically in Xcode 4. This is a very powerful time-saving feature.



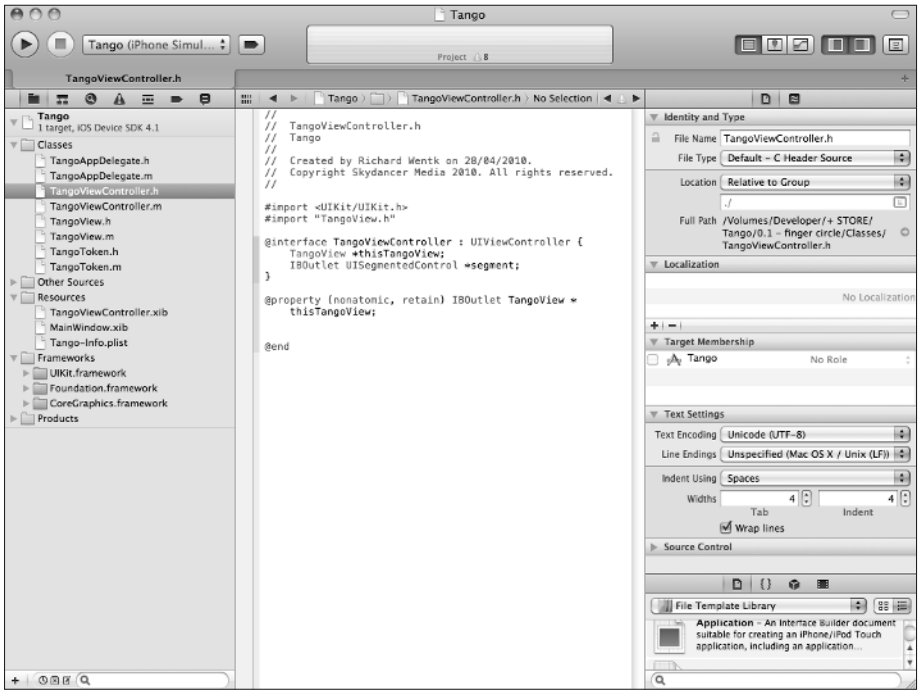
Exploring code and file management

Xcode 4 includes two new 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. As shown in Figure 1.11, the left pane, known as the *navigation pane*, includes a simplified but familiar version of the Groups & Files pane from Xcode 3. This pane also includes symbol lists, search options, and log listings.

The Get Info feature in Xcode 3 has been replaced by an info view in Xcode 4. As shown in Figure 1.11, you can display the filename, type, and location in a right pane, which is known as the *utilities pane*. This pane gathers miscellaneous information that previously appeared in various floating windows. For example, IB's inspector panes appear here. It also shows build target and localization information. When you select a file, the contents of this pane are updated automatically. This saves time over Xcode 3's Get Info pop-up menu feature, which presented this information in a less accessible way.

Figure 1.11

New left and right panes in Xcode 4 display ancillary information and manage optional features that may not be needed while editing.

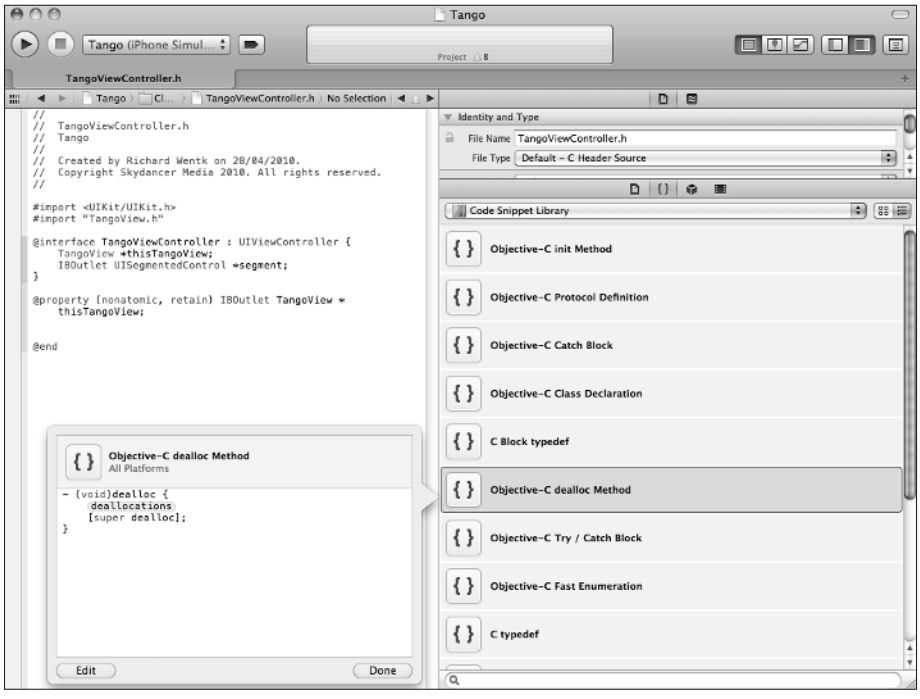


The utilities pane is also shown in Figure 1.11. It includes a new Library sub-pane that can display file templates, standard code snippets, standard system objects that include both UI and data classes, and project media files.

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 previews 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 new Code Snippets feature makes it easy to reuse code and is a partial replacement for Xcode 3's Code Sense macros.

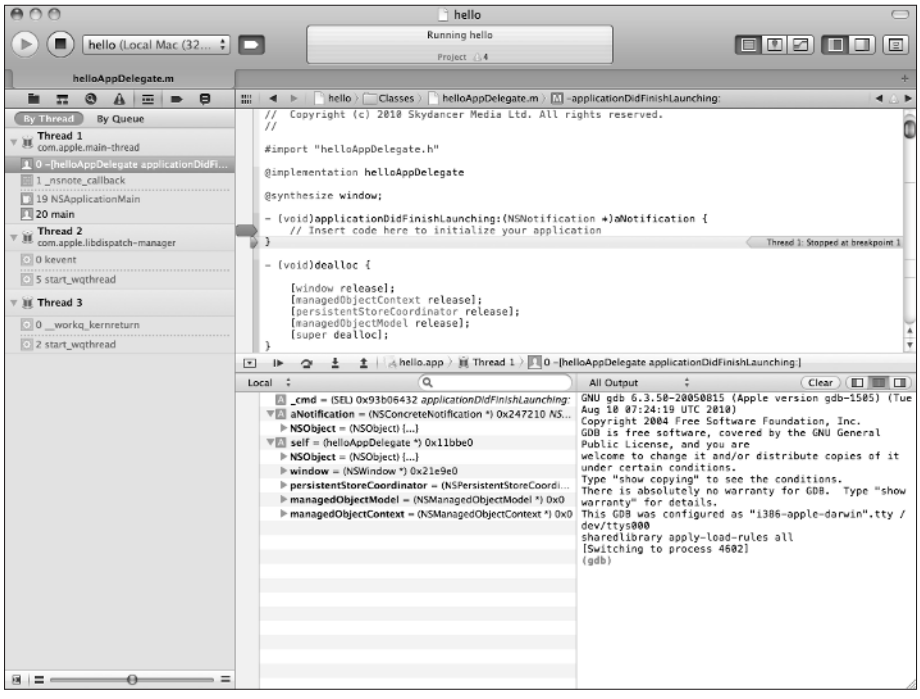


Exploring the debugger

As shown in Figure 1.13, the debugger now 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 new debugger area no longer appears in a separate window, although for convenience you may decide to launch it in one. 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 4 is build-and-go. There are no restrictions on development, testing, or distribution. 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. You can also create Mac apps for the App Store—but this is one development option, and not an obligation. Figure 1.14 shows a simple OS X application using a template as a starting point.

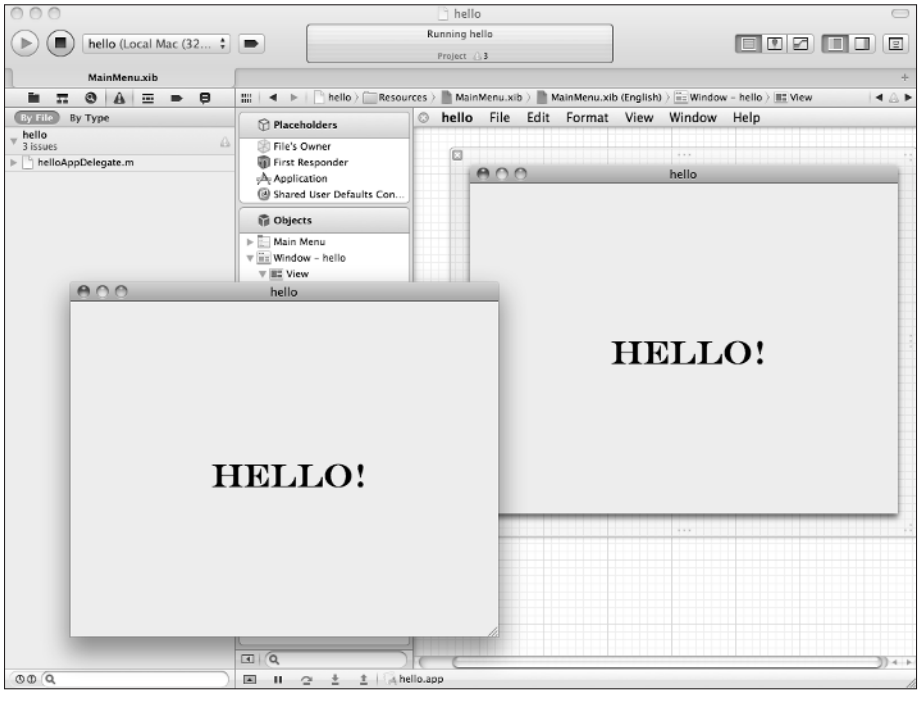
NOTE

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

Earlier versions of Xcode supported universal binary development, which was backward-compatible with PowerPC hardware. Although OS X 10.6 Snow Leopard was the first Intel-only version of OS X, Xcode supports universal binaries for Mac development and can still be used to develop applications for PowerPC targets. It continues to support AltiVec hardware acceleration. The commercial market for PowerPC applications is tiny, but PowerPC applications remain interesting in specialized media and scientific computing.

Figure 1.14

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.



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 very device.

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.

Apple is unlikely to simplify this process in future releases of Xcode. It's more likely that the iOS device range will expand, and apps will need to manage an ever-increasing selection of screen sizes, hardware features, and operating systems.

The Xcode Simulator, shown in Figure 1.15, includes separate iPhone and iPad testing options, but it is suitable only for apps that don't use any of the iOS hardware features, such as the GPS and the accelerometer. All but the very simplest commercial apps should be tested on real hardware.

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.

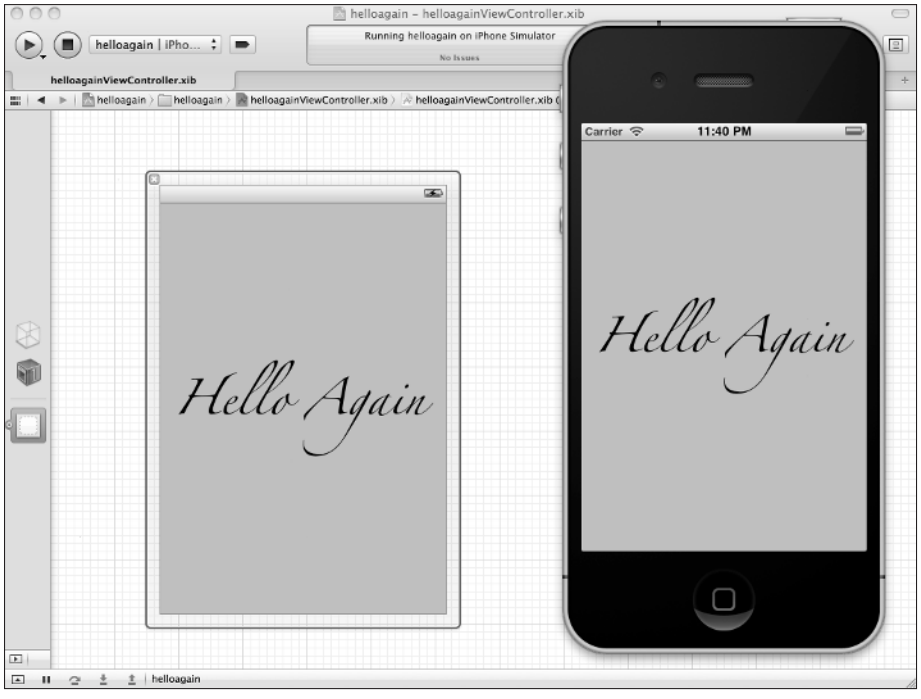
To date, Apple's beta cycle for iOS has been more aggressive than for OS X, and it's usual to have a new beta preview of iOS available almost as soon as the most recent version has been made public. Beta development requires a parallel version of Xcode and beta firmware for every test device. It may 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 very much simpler, you should allow extra time to work for projects to support all the different possible targets, versions, and security options.

Xcode supports these extra possibilities, but it doesn't simplify them. Developing and testing a universal iOS app—a single app that can run on an iPhone, an iPod, and an iPad—remains a challenge.

Figure 1.15

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.



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 core programming model:** iOS supports task switching with very limited multi-tasking, and limited access to the device file system. OS X supports a wider range of multi-tasking options, and less restricted access to files.
- **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 sketches 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 remain possible but difficult. It's more realistic to think of Xcode 4 as two separate development environments with a common frontend than as a single unified environment designed to produce code for either platform.

Summary

This chapter explained how previous Mac development tools led to Xcode, listed the key design goals of Xcode 4, and summarized the key differences between Xcode 4 and Xcode 3. It introduced some of the practical differences in more detail and looked at some of the new features. Finally, it explored the fundamental differences between iOS and OS X development and briefly discussed how those differences affect the development workflow.