Chapter 1

Xcode Projects

Every program you create in Xcode requires a project, even a simple command-line program with one file. Because every program requires a project, covering projects is a good way to start the book. After reading this chapter you will know how to create a project and add files to a project. You will also become familiar with Xcode's project window.

Creating a Project

For most of you, the first you'll do when you start with Xcode is create a project. An Xcode project contains your program's source code files and other files, such as xib files, Xcode needs to build a working program. When you launch Xcode, a welcome window opens that gives you the option of creating a new project. You can also choose File > New > Project to create a new project. Creating a project is a three-step process.

Step 1: Choose the Type of Project You Want to Create

When you create a new project, the New Project Assistant opens, which you can see in Figure 1.1. On the left side of the window is a list of project categories. Xcode has the following project categories:

- Application
- Framework and Library
- Application Plug-in
- System Plug-in
- Other
- iOS Application
- iOS Framework and Library
- iOS Other, which lets you create an empty project or in-app purchase content.

Categories   Project Templates

Figure 1.1

Descriptions

New Project Assistant

I will go into greater detail on the types of projects shortly, but most of you will be making application projects. Selecting a category shows the projects you can create for a given category. When you select a project, a description appears at the bottom of the New Project Assistant. After choosing the type of project you want to make, click the Next button to move on to Step 2.

Step 2: Choose a Product Name

What you can do in Step 2 depends on the type of project you select in Step 1, but choosing a product name is one thing you will do for every project. The product name is the name of what Xcode builds. For an application the product name is the application name. The product name is also the name Xcode gives the project file.

Xcode 4.4 adds an Organization Name text field for most project templates. The name you enter in the text field appears in the copyright notice at the top of new source code files Xcode adds to the project.

Many Xcode project templates let you specify a company identifier along with the product name. The company identifier uniquely identifies you as the creator of the product. It takes the following form:

```
com.CompanyName
```

You can use your name as a company name if you don't work for a company.

If your project uses the Cocoa or Cocoa Touch frameworks, you will see a checkbox named Include Unit Tests. Selecting this checkbox tells Xcode to add a unit testing bundle target to your project. Select the checkbox if you're going to unit test your project.

Projects that use the Cocoa and Cocoa Touch frameworks should see a Use Automatic Reference Counting checkbox. Selecting the checkbox tells Xcode to use automatic reference counting, which simplifies memory management in Objective-C applications. I recommend using automatic reference counting for new projects.

Cocoa and iOS applications may see a Class Prefix text field. If you enter a prefix in this field, Xcode adds the prefix to any classes and class files you create in the project. An example of a class prefix is the NS prefix used in many Cocoa classes. If you enter `XYZ` for a Cocoa application project, Xcode adds the files `XYZAppDelegate.h` and `XYZAppDelegate.m` to the project. The name of the class in this case `XYZAppDelegate`.

As I said at the beginning of this section, the choices you can make in Step 2 depend on the type of project you create. Some examples of the choices you can make include the following:

- Using Core Data in Cocoa applications and some iOS applications.
- The language to use for a command-line tool.
- The type of library to create for a library project: static or dynamic.
- Creating a document-based Cocoa application.
- Picking the device for some iOS applications: iPhone or iPad.

After making your choices, click the Next button to move on to the final step.

Step 3: Save the Project

Tell Xcode where you want to store the project and click the Create button. Congratulations! You've created an Xcode project.

At the bottom of the Save panel you will see a checkbox named Create local git repository for this project. If you select this checkbox, Xcode creates a git repository for your project and places the project under version control. A version control system tracks the changes made to files and who made the changes. Read Chapter 8, "Version Control", for more information on version control.

If you're learning programming, you don't need to create a git repository for your project. Larger projects and projects with a long lifespan are the types of projects that would benefit from creating a git repository.

The Project's Contents

After creating a project, you should see its contents on the left side of the project window. What Xcode includes in a project depends on the type of project you create. Xcode includes the following files for a Cocoa application:

- Three Objective-C source code files: `main.m`, `AppDelegate.h`, and `AppDelegate.m`. The app delegate files may have the name of your project as a prefix.
- The Cocoa framework.
- A xib file, `MainMenu.xib`, containing the user interface. Select the xib file to build the user interface.
- Two property list files, `Info.plist` and `InfoPlist.strings`. The property list files may have the name of your project as a prefix, like `ProjectName-Info.plist`. Property list files are XML files that consist of key/value pairs. The key stores the property name, and the value stores the property's value. The `Info.plist` file contains information about the application, such as its name and version number. The `InfoPlist.strings` file contains localized application information, such as a copyright notice. There will be one `InfoPlist.strings` file for each spoken language your application supports.

- A prefix header, `ProjectName-Prefix.pch`. Xcode precompiles the header files in the prefix file, which makes your project compile faster.
- A rich text file, `Credits.rtf`, that lets you credit everyone who helped create the application.

Each project type includes its own set of files. A Core Data application is going to use the Core Data framework along with the Cocoa framework. It also will include a data model file with the extension `.xcdatamodeld`. A C++ command-line application won't include any frameworks or xib files. What's important to know is each of Xcode's project types provides a starting point for writing your own programs. You can focus on writing code rather than worrying about forgetting a framework.

Assuming you choose a Cocoa application project, you can run it without writing any additional code. If you're getting antsy to do something on the computer, you can test my previous statement by clicking the Run button on the left side of the project window toolbar. Xcode will compile the source code and run the application, which displays a blank window on the screen.

Application Projects

Application projects enable you to create Mac applications that can be launched from the Finder. You can create GUI applications using the Cocoa framework or command-line applications that run in the Terminal application. If you're learning programming or learning Mac development with Xcode, you'll be creating application projects.

Cocoa Projects

You have a choice of two Cocoa projects: Cocoa Application and Cocoa AppleScript Application. The Cocoa application project uses Objective-C as its programming language. Objective-C is Cocoa's native language and is the language to use if you're not sure about what language to use.

As its name implies, the Cocoa AppleScript application project lets you write a Cocoa application in AppleScript. AppleScript is good for small applications, but it's not suited for large applications. AppleScript was designed for writing scripts to automate other applications, not for writing large applications.

Document and Non-Document Applications

Cocoa applications give you the option of writing an application or a document-based application. The difference between the two application types is document-based applications can have multiple documents open at one time. Text editors, spreadsheets, and web browsers are examples of document-based applications; you can have five documents open at once. iPhoto is an example of an application; you don't open five windows with each one displaying a different photo. If your application creates a new window when the user chooses File > New, you should create a document-based application.

If you create a document-based Cocoa application, Xcode asks you for the name of the document class, which is a subclass of `NSDocument`. Xcode creates a source code file and a xib file with the document class name you supply. Xcode also asks you for a file extension for the documents your application creates.

Core Data Applications

Core Data application projects are Cocoa applications that use the Core Data framework. The Core Data framework makes creating data structures for your Cocoa applications easier. Core Data lets you use Xcode's modeling tools to define your program's data structures instead of writing code. Read Chapter 5, "Modeling Tools", for more information on Xcode's modeling tools.

If you decide to use Core Data in a Cocoa application project, you have the option to include a Spotlight importer. Your application would require a Spotlight importer if it uses a custom document format and you want Spotlight to be able to find your application's files when the user searches in the Finder.

Command-Line Tool Projects

Command-line tool projects create programs that run from the Terminal application instead of the Finder. If you're learning C, C++, or Objective-C, command-line tool projects are perfect. They let you use the standard C and C++ functions to print output on the screen. Those functions don't work with graphical user interfaces like Mac OS X's. By using a command-line tool project, you can learn the language without having to deal with the complexity of writing Mac programs. You can write command-line tool programs in C, C++, or Objective-C. Use the Type pop-up menu to choose the language. Choose Foundation to create an Objective-C program.

You can also create command-line tool projects that use the Core Data, Core Services, and Core Foundation frameworks. Most Core Data applications work with the Cocoa framework, but you can use Core Data to write a command-line program.

Core Services contains the operating system services that have nothing to do with a program's user interface, such as networking, file management, memory management, and multiprocessing. Higher-level frameworks like Cocoa sit on top of Core Services. The Core Foundation framework is the C language equivalent of the Foundation framework. It allows C and C++ programs to take advantage of the Foundation framework.

Framework and Library Projects

The projects in the Framework and Library group are meant for reusable code. The code could be used by you in multiple projects or it could be used by developers all over the world in their projects. There are three categories of projects in the Framework and Library group: libraries, frameworks, and bundles.

Libraries

You can write libraries in C, C++, or Objective-C. If your library is going to be used on operating systems other than Mac OS X, use a C or C++ library project.

When you create a library project, you must choose between creating a static or a dynamic library. A *static library* is linked to an application when compiling the application. A *dynamic library* is linked when the application runs. Static libraries are easier to use, but dynamic libraries can save memory if multiple running applications use the same library. Suppose you have a library that five running applications use. A static library would load five copies of the library, one for each application. A dynamic library would load one copy, shared by the five running applications.

Frameworks

Frameworks are similar to dynamic libraries, but they also contain header files and resource files. Frameworks are easier to use than dynamic libraries. The problem with frameworks is they work only on Mac OS X. Frameworks are a good choice for reusing Cocoa code.

Bundles and XPC Services

You use bundle projects most often to create plug-ins, programs other applications use to add features to the application. Lots of applications support plug-ins, such as iPhoto, Photoshop, Maya, and REAL Studio. Bundles can be created using the Cocoa and Core Foundation frameworks.

An XPC service provides a low-level method of interprocess communication that is compatible with Apple's Grand Central Dispatch technology. Apple's *Daemons and Services Programming Guide*, which is part of Apple's developer documentation, has more information on XPC services.

Application Plug-in Projects

The Application Plug-in group has project templates to create plug-ins for Automator, Address Book, Installer, and Quartz Composer. The Automator plug-in project lets you create Automator actions. An action is a loadable bundle that performs one task. Create workflows in Automator by linking actions together. Use workflows to automate tasks in your application and other applications. You can write Automator actions using AppleScript, Objective-C, or shell scripts.

System Plug-in Projects

The system plug-in projects are lower level than the application plug-in projects.

- Generic C++ plug-in
- Generic kernel extension
- Image unit plug-in
- IOKit driver
- Preference pane
- Quick Look plug-in
- Screen saver
- Spotlight plug-in
- Sync schema (removed in Xcode 4.4)

Use the generic C++ plug-in project if you need to write a plug-in that isn't covered by the other projects in the System Plug-in group.

The generic kernel extension project lets you write a kernel extension. Kernel extension programming is low-level operating system programming with the potential to wreck your computer so be careful if you choose to write a kernel extension.

The image unit plug-in project creates a plug-in for Core Image. Core Image is a Mac OS X technology for applying effects and filters to images.

Use the IOKit driver project to write drivers for peripherals like joysticks, gamepads, printers, and scanners.

The preference pane project lets you build a preference pane, which is a plug-in for the System Preferences application. Choose Apple > System Preferences to launch System Preferences.

If you have a custom document type, a Quick Look plug-in for that document type lets the user examine a document by choosing File > Quick Look in the Finder.

The screen saver project lets you write a Mac screen saver.

The Spotlight plug-in project is used to import metadata from a file. If your application uses a custom file type and you want it to be searchable in the Finder using Spotlight, create a Spotlight plug-in.

The sync schema project lets you build a schema to synchronize data between your application, other applications, and other devices. Suppose you have an application the user has installed on a desktop Mac. With a sync schema the user could synchronize their data with a laptop, cell phone, or iPod.

Other Projects

There are three possible templates in the Other group: empty project, external build system project, and in-app purchase content project. External build system projects use a program other than Xcode to build the projects. There are two cases where you would use an external build system project. The first case is when you want to use a different build system than Xcode's to build your project. This situation occurs most often when you're writing software for multiple operating systems. Build systems like `make`, SCons, and CMake let you build your program in a cross-platform manner. You can use one of these build systems to build a Mac application in Xcode, a Windows application in Visual Studio, and a Linux application in Eclipse.

The second case to use an external build system project is when you're writing a program in a language other than the ones for which Xcode supplies compilers: AppleScript, C, C++, and Objective-C. An external build system project lets you use Xcode to write code in any programming language, as long as you install that language's compiler on your Mac. Mac OS X ships with Java, Perl, Python, and Ruby so you don't have to install anything to use those languages.

When you choose to create an external build system project, Xcode asks you for the path to the build tool you're going to use to build the project. The initial tool is `make`. If you're going to use a different tool to build your project, enter the path to the tool in the Build Tool text field.

Xcode 4.4 adds the in-app purchase content project template for both Mac and iOS applications. Because in-app content requires an app, most of you will add an in-app purchase content target to an application project instead of creating a new project. Refer to the "Adding Targets" section in Chapter 6, "Building Projects", for more information on adding targets to a project.

iOS Application Projects

The iOS Application group has project templates to write applications that run on the iPhone, iPod Touch, and iPad. Xcode has the following iOS application templates:

- Master-detail application
- OpenGL game
- Page-based application
- Single view application
- Tabbed application
- Utility application
- Empty application

Master-detail applications usually contain a master list of items. Selecting an item from the list displays information about the selected item. The master-detail application template includes a navigation controller to display the master list of items. The template also includes a split view for iPad applications.

The OpenGL game template includes the OpenGL ES framework. If you're writing a game or an application that requires high-performance graphics, use the OpenGL game template.

The page-based application template includes a page view controller, which simplifies dealing with multiple pages of information in your application. The single view application template is for iOS applications that use only one view.

The tabbed application template includes a tab bar and a view controller for the tab bar. If your application uses a tab bar, choose the tabbed application template.
The utility application template is used to write simple applications that do not require much input from the user. An example of a utility application is an application that provides a five-day weather forecast.

The empty application template provides a window and an application delegate. You can use this template as the starting point for any iOS application.

Devices

When you choose a product name for your iOS application project, you should see a Devices pop-up menu. The menu determines the devices your application supports and determines the xib file Xcode creates for you. If you pick iPad, Xcode gives you an iPad xib that includes an iPad-sized main window. If you pick iPhone, Xcode gives you an iPhone xib that includes an iPhone-sized main window. If you pick Universal, Xcode creates two xib files: one for iPhone and one for iPad.

Creating a universal application is easy. All of the iOS application templates can create a universal application that runs on both iPhones and iPads. Choose Universal from the Devices menu.

Core Data

Several of the iOS application templates give you the option to use Core Data. Core Data lets you use Xcode's modeling tools to define your program's data structures instead of writing code. If you can use Core Data, there will be a Use Core Data checkbox below the Product Name text field. Select the checkbox to use Core Data. If you decide to use Core Data, Xcode adds the Core Data framework and a data model to the project.

Storyboarding

When naming your project, you should see a Use Storyboards checkbox. Selecting the checkbox tells Xcode to add storyboard files to your project instead of xib files. Storyboards allow you to view all your application's screens in one place. Read the section "Storyboarding" in Chapter 4, "Creating User Interfaces for iOS Applications", for more information on storyboarding.

iOS Library Projects

The library project template lets you create a static library using the Foundation framework.

Project Window

The project window, shown in Figure 1.2, is your project's home when working in Xcode. It has the following components:

- Toolbar
- Navigator
- Editor
- Utility area
- Debug area

All five areas of the project window may not be visible initially. Use the View menu or the View buttons on the project window toolbar to control what appears in the project window. The View menu also lets you go to specific areas of the project window, such as going to a specific navigator.

Toolbar

At the top of the project window is the toolbar, which provides you easy access to tasks you perform most. On the left side of the toolbar are controls related to running your program in Xcode. There are buttons to run your program and stop running it. Next to the Run and Stop buttons is the Scheme menu. A scheme lets you control how Xcode builds, tests, and launches your project. I cover schemes in more detail in Chapter 6, "Building Projects"; read the "Schemes" section.
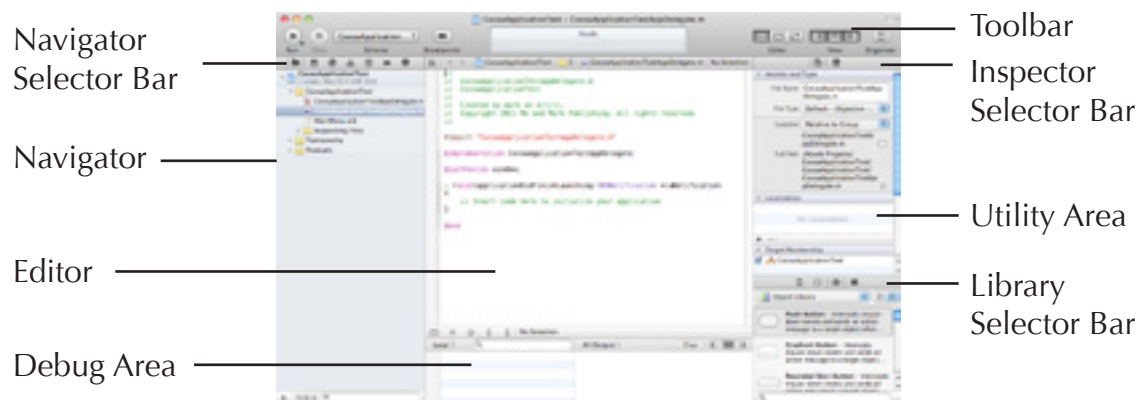

Figure 1.2

Project window

Next to the Scheme menu is a Breakpoints button that toggles your breakpoints. Initially any breakpoints you set are deactivated. Clicking the Breakpoints button activates the breakpoints you set, and Xcode runs your project in the debugger when you click the Run button. Clicking the button a second time deactivates the breakpoints, and Xcode runs your project outside the debugger. Chapter 7, "Debugging", has more information on debugging and breakpoints.

In the center of the toolbar is the status area. It shows the progress of time-consuming tasks like downloading documentation.

On the right side of the toolbar are three sets of buttons: Editor, View, and Organizer. The Editor group has three buttons and tells Xcode what editor to use: standard, assistant, or version. The standard editor gives you one large area for editing. The assistant editor splits the editor in two. For source code files the assistant editor shows the file and its counterpart. If you select a header file in the navigator, the assistant editor shows both the header file and its corresponding implementation file. The version editor works only with files under version control. It lets you compare versions of a file. You can also use the View menu to pick your editor by choosing View > Standard/Assistant/Version Editor > Show Standard/Assistant/Version Editor.

The View group of buttons determines what project views are visible. The first button toggles the navigator. The second button toggles the debug area. The third button toggles the utility area. You can also use the View menu to hide and show the navigator, debug area, and utility area.

The Organizer group isn't really a group because it has only one button. Clicking it opens the Organizer window. Read the "Organizer" section later in this chapter for more information on the Organizer.

Navigator

Xcode projects have a lot of information to track. The information you want to access depends on what you're currently doing. When you're writing code, you want access to the files in the project. When you build your project, you want access to compiler errors and build logs. When debugging, you want access to the breakpoints you set. The navigator lets you quickly access different areas of your project.

Above the navigator is the navigator selector bar. The navigator selector bar has buttons to pick the navigator to show. The selector bar has the following buttons, running from left to right:

- Project navigator
- Symbol navigator
- Search navigator
- Issue navigator
- Debug navigator
- Breakpoint navigator
- Log navigator

You can also go to a specific navigator by choosing View > Navigators.

Project Navigator

When you create a new project, the project navigator is the navigator you see. The project navigator shows the files in your project. Selecting a file from the project navigator opens the file in the editor.

The project navigator initially shows all the files in the project. At the bottom of the navigator are controls to filter what appears in the project navigator. The search field lets you enter text. If you enter `.m` in the search field, the project navigator shows all the Objective-C implementation files in your project.

Next to the search field are three tiny buttons. Clicking the first button tells Xcode to show recently edited files in the project navigator. Clicking the second button tells Xcode to show files with source control status, files that are different from the version in the version control repository. If your project is not under version control, clicking the second button hides all the files in the project navigator. Clicking the third button tells Xcode to show only files with unsaved changes.

Project Navigator Groups

When you examine your project's contents in the project navigator, you should see some folders. These folders are groups, which help you organize your project's files in Xcode. Groups do not correspond to folders in the file system; they exist only in Xcode.

You can create your own groups by right-clicking in the project navigator and choosing New Group. If the New Group menu item is disabled, make sure you right-click on a file or folder. The menu item is disabled if you right-click below the project's files. Selecting some files, right-clicking, and choosing New Group from Selection creates a new group and places the selected files in the group. Dragging a file from the project navigator to the folder adds the file to the group.

Select the group in the project navigator and either click or press the Return key to rename the group. Select the group and press the Delete key to remove the group. If you have files inside the group, deleting the group also deletes the files in the group. If you don't want to delete the files, drag them out of the group before you delete the group.

Symbol Navigator

The symbol navigator allows you to view the symbols in your projects. Examples of symbols are classes, variables, functions, data structures, and enumerated data types. The symbol navigator has the following groups, shown in Table 1.1:

Xcode initially shows only classes in the symbol navigator. To see the other groups, go to the bottom of the symbol navigator. There is a group of three tiny buttons next to the search field. Click the left button, the one with the letter C, to show the other groups.

For each group listed in Table 1.1, Xcode tells you the number of that group defined in the project and the number defined in the system. Examples of system-defined symbols are symbols from the Cocoa framework and symbols from the Standard C Library. Normally the number of system-defined symbols is larger than the number of project-defined symbols. A small Cocoa application might have 5-10 classes in the project and have several hundred system classes because the Cocoa framework is large.

Table 1.1 Symbol Navigator Groups

Symbol Type	Description
Classes	Classes and their members. The Classes group is the one Objective-C and C++ developers will use the most.
Protocols	Objective-C protocols. A protocol is a list of methods for another class to implement.
Functions	Functions that are not member functions of a class, such as C functions.
Structs	C structures.
Unions	C unions, which are similar to structs, but only one field of the data structure is stored in memory at one time.
Enums	Enumerated data types that use the `enum` statement.
Types	User-defined data types. If you use the `typedef` statement to define data types, they appear under Types.
Globals	Global variables.

Click the disclosure triangle next to an entry to see its contents. Xcode initially shows only project-defined symbols. Selecting a symbol from the symbol navigator opens the symbol's file in the editor and takes you to the symbol.

There are two buttons above the symbol navigator: Hierarchical and Flat. Clicking the Hierarchical button shows a hierarchical symbol listing. Clicking the Flat button provides one flat list of symbols in alphabetical order.

Filtering the Symbol Navigator's Contents

There are two ways to filter what appears in the symbol navigator. First, you can enter text in the search field. Symbols matching the text you entered appear in the symbol navigator.

Second, you can use the three tiny buttons next to the search field to filter. The buttons from left to right are the following:

- Show only class symbols. Clicking this button shows a list of classes and hides the other groups. Showing only class symbols can help for Objective-C applications that don't have many entries for anything besides classes.
- Show only project-defined symbols. Clicking this button hides system-defined symbols. Showing only project-defined symbols allows you to focus on the symbols in your code. It also improves symbol navigator performance because Xcode doesn't have to show the system-defined symbols.
- Show containers only. Clicking this button affects the Classes group. Showing containers lists each class in the symbol navigator, but does not show the members of each class.

Search Navigator

The search navigator is used for project-wide search. Choose Edit > Find to search a single file. Start entering your search term in the search field. A menu opens that has two sections: search in project and search in project and frameworks. In both sections you can find text containing the search term or find text starting with the search term. Finish entering the search term and either choose an option from the menu or press the Return key.

Xcode groups the search results by file. Selecting a search result opens the file in the editor and takes you to the result.

Customizing Your Search

Clicking the magnifying glass button in the search field and choosing Show Find Options brings up options in the search navigator for you to customize your search. You have the following options:

- Style: textual or regular expression. A regular expression is a string that describes a search pattern. If you are unfamiliar with regular expressions, use a textual search.
- Hits must: contain, start with, match, or end with search term.
- Case: ignore or match. Suppose you enter the search term `string`. If you ignore the case the search navigator displays results for `string`, `String`, and `STRING`. If you match case, the search navigator displays results for `string` only.
- Find in: workspace or custom. The "Find Scopes" section shows how to create a custom find.
- Checkbox to search linked frameworks. Searching linked frameworks can bring up lots of results.

Xcode 4.4 adds two styles for searching your project's symbols: symbol definitions and symbol references. When you use symbol definitions, the search navigator generates a listing for each place in your code that defines the symbol. When you use symbol references, the search navigator generates a listing for each place in your code that refers to the symbol. Symbol references generate more listings than symbol definitions.

Suppose you search for a function in your project named `DoSomething`. If you search symbol definitions, the search navigator displays one listing where you defined `DoSomething` in the header file and another listing where you wrote the code for `DoSomething` in the implementation file. If you search symbol references, the search navigator creates the two listings it creates for symbol definitions, plus a listing for each place in your code that calls `DoSomething`.

Xcode 4.4 adds the ability to insert a pattern into the search field. Click on the magnifying glass icon in the search field and choose Insert Pattern. Inserting a pattern allows you to search for things like tabs, white space, hexadecimal digits, email addresses, and URLs. You can add patterns to a search term to fine tune the search results.

Find Scopes

Create a find scope when searching in the workspace is not good enough. To create a find scope, choose Custom from the Find in menu in the search navigator.

When you choose Custom from the Find in menu, a sheet opens where you create find scopes. Figure 1.3 shows an example of the find scope sheet. On the left side is a list of find scopes. The list is empty if this is the first find scope you're creating. Click the + button to add a find scope. Double-click a scope entry to change its name. The scopes you create end up in the Find in menu.

Selecting a find scope from the list displays a list of conditions that must be met to show up in the search results. Click the + button to add a condition. The find scope editor has the following conditions for you to add:

- Location, which lets you search files based on location: within workspace or within a file or folder. Searching within the workspace only makes sense if you're combining other conditions because you can search within the workspace without creating a find scope. Searching within a folder would help if you wanted to search in a subfolder of your project. If you choose to search within a file or folder, click the Choose Path pop-up cell to pick a file or folder.
- Name, which lets you search files based on file name. Searching by name has the following conditions: is equal to, is not equal to, contains, starts with, ends with, or matches regex. Regex is a regular expression. Enter the term in the text field.
- Path, which lets you search multiple paths. A path has the same conditions as Name.
- Path Extension, which lets you search files based on file extension. A path extension has the same conditions as Name and Path.
- Type, which lets you search files based on file type. The Type condition has two pop-up cells. The first cell lets you specify is or is not. The second cell lets you specify the type of file: source code, script, a specific language, HTML, XML, any, or other type. Choose Other Type to specify a file type not listed in the pop-up cell. Enter the file extension in the text field.

Find and Replace

To do find and replace, click the Find menu next to the search field and choose Replace. An unlabeled text field appears below the search field along with Preview, Replace, and Replace All buttons. Enter the replace term in the text field. The navigator shows the search results.

Clicking the Preview button lets you preview the changes, which you can see in Figure 1.4. In the preview window there is a listing for each search match in the project with a checkbox next to it. Selecting the checkbox tells Xcode to change that instance. Next to the listings are two panes. The left pane shows what the code will look like if you replace the text. The right pane shows the current code. There is a switch between the two code fragments with an indicator. The indicator pointing left tells Xcode to change that instance. Click the Replace button to make the changes.

Issue Navigator

The issue navigator lists any compiler errors, warnings, and static analyzer issues in your project. The sections "Seeing More Build Details" and "Static Analysis" in Chapter 6, "Building Projects", have more information on the issue navigator.

Debug Navigator

The debug navigator lists the call stack when debugging your program. Read the "Debug Navigator" section in Chapter 7, "Debugging", for more information on the debug navigator.

Breakpoint Navigator

The breakpoint navigator lists the breakpoints you have set in your project. Read the section "Setting Breakpoints" in Chapter 7, "Debugging", for more information on the breakpoint navigator.

Log Navigator

The log navigator lets you view the logs of tasks you perform on a project, such as building the project, running it, and committing files to a version control repository. Select a log from the list to open it in the editor.

Figure 1.3

Find scope sheet

Figure 1.4

Find and replace preview

Editor

What you do in the editor depends on the file you select in the project navigator. If you select a source code file, the editor lets you edit the source code. If you select a xib file, you can modify your project's user interface. If you select the project file, you can edit project and target settings.

As I explained in the "Toolbar" section, Xcode has three editors: standard, assistant, and version. The standard editor provides one editing area and is the right editor to use to edit non-source code files. The assistant editor provides multiple editing areas. It is most useful for editing source code files and making connections to your code in xib files. The version editor compares two versions of a file. Your project must be under version control to use the version editor.

I cover the assistant editor in more detail in Chapter 2, "Editing Source Code", specifically in the "Assistant Editor" section. I cover the version editor in more detail in Chapter 8, "Version Control", specifically in the "Seeing the Changes You Made to a File" section.

Utility Area

The utility area contains inspectors and libraries. Inspectors let you tweak settings for a file. Use the inspector selector bar (see Figure 1.2) to pick an inspector. The inspectors that are available depend on the file type, but every file has at least two inspectors: File and Quick Help.

File Inspector

The file inspector shows information about a file. Select a file from the project navigator to see its information in the file inspector. The file inspector has the following sections for text files: Identity and Type, Localization, Target Membership, Text Settings, and Source Control.

Identity and Type

The Identity and Type section lets you specify a file's name, type, and location. The Location pop-up menu determines how Xcode stores the file's location. There are six options.

- Relative to group, which means Xcode uses the file's group in the project navigator to store the file's location.
- Relative to project, which means Xcode uses the project's folder to store the file's location.
- Absolute path, which means Xcode uses the file's path on your computer to store the file's location.
- Relative to build products, which means Xcode uses the folder where it places build products, such as executable files and libraries, to store the file's location.
- Relative to Developer directory, which means Xcode uses the folder where you installed Xcode to store the file's location.
- Relative to SDK, which means Xcode uses the folder where you installed the current SDK to store the file's location.

The default location type for files is by group. The reference type options matter if you're going to share your projects with other people. If you refer to your files according to the absolute path on your computer, chances are high that other people using your project have a different path to the files. In this case Xcode would be unable to find the files and the project would not compile. In most cases you should choose either Relative to Group or Relative to Project from the Location pop-up menu.

Localization

The Localization section lets you add a localization for a file. This section will be blank for source code files. You shouldn't have to add a localization for source code files. Files you would need to localize have one localization for the language you're using, which is English for most of you.

Target Membership

The Target Membership section tells you the targets the file belongs to. If you need a file to be a member of the target, select the checkbox next to the target.

Source code files, xib files, and files that are copied to the application bundle are the files in your project that are members of targets. Header files are normally not target members.

Text Settings

The Text Settings section let you control the text encoding, line endings, indentation, and line wrapping in the editor. In most cases you should use Xcode's Text Editing preferences to control these settings. The Text Settings section of the file inspector lets you control the text settings for a single file or for a single project. Select the project file from the project navigator to make changes to the text settings for the project. The section "Text Editing Preferences" in Chapter 2, "Editing Source Code", provides more information on Xcode's Text Editing preferences.

Source Control

Source control tells you the version of the file and its source control status. If the file's status is modified, there is a Discard button that lets you discard the changes you made to the file.

Your project must be under version control to have a Source Control section. The section "Seeing Which Files Have Changed in Your Project" in Chapter 8, "Version Control", provides more information on source control status.

Quick Help Inspector

If you select some text in the editor, the Quick Help inspector shows the Quick Help documentation for the selected text. For Cocoa and Cocoa Touch classes, the Quick Help documentation provides an overview of the class or method as well as links to documentation. The "Quick Help" section in Chapter 2, "Editing Source Code", contains more information on Quick Help.

Library

Below the inspectors is the library. The library has content you can add to your project. Use the library selector bar (see Figure 1.2) to pick a library. There are four libraries: file template, code snippet, object, and media.

The file template library contains templates for blank files you can add to your project. Select a file from the library and drag it to the project navigator to add the file to your project. Read the section "Creating New Files for the Project" later in this chapter for information on the files you can add to your project.

The code snippet library contains code snippets. Drag a snippet from the library to the editor to add the snippet to your code. Read the "Code Snippets" section in Chapter 2, "Editing Source Code", for more information on the code snippet library.

The object library contains user interface elements. Use the object library to build your application's user interface. Read the "Object Library" section in Chapter 3, "Creating User Interfaces for Mac Applications", for additional information on Mac user interface elements. Read the "Object Library" section in Chapter 4, "Creating User Interfaces for iOS Applications", for more information on iOS user interface elements.

The media library contains graphics, audio, and video files. Use the media library to play audio and video files and to view graphics files. To have entries in the media library, you must either add media files to your project or open a xib file in a Mac project. Read the "Media Library" section in Chapter 3, "Creating User Interfaces for Mac Applications", for more information on the media library.

Debug Area

The debug area of the project window becomes important when debugging your program. It shows your program's variables, the debug console, and controls for controlling the execution of your program. I cover the debug area in more detail in Chapter 7, "Debugging".

Adding Files and Frameworks to Your Project

Xcode includes a source code file when you create a new project, but unless you're writing a very simple program, you're going to be adding files to your project. Examples of files you add to a project include source code files, xib files, data files, graphics files, and audio files. You can create new files and add existing files to your project.

Creating New Files for the Project

To create a new file and add it to your project, choose File > New > File or drag a file from the file template library to the project navigator. Creating a new file consists of two steps: choosing a file type and naming the file. Dragging a file from the file template library to the project navigator is the equivalent of choosing a file type.

Choosing a File Type

When you create a new file by choosing File > New > File, the New File Assistant opens, which you can see in Figure 1.5. The left side of the assistant contains a list of template categories. Selecting a category fills the sheet with the category's templates. Selecting a file type from the list displays a description of the file type.

Select the type of file you want to create from the list. Click the Next button to move on to naming the file. Some file types have an intermediate step before naming the file. When you add an Objective-C class, you specify the new class's superclass before naming the file.

Naming the File

After selecting the type of file you want to create, click the Next button, which will take you to the final part of the file creation process. Name your file. Notice that Xcode automatically includes the appropriate ending depending on the type of file you create. A C file will have the extension **.c**.

If you create an Objective-C file, you don't explicitly name it. Suppose you add an Objective-C class to your project. The New File Assistant has a Class text field. The class name you enter is the name of the file.

After naming the file tell Xcode where you want the file to reside on your hard disk. By default it will be in the same folder where your project is. You also have the options to determine which group in your project to place the file and determine the targets in your project where the file should be added.

Click the Save button to create the file. At the top of the file is a comment containing the following information:

- The file name.
- The project name.
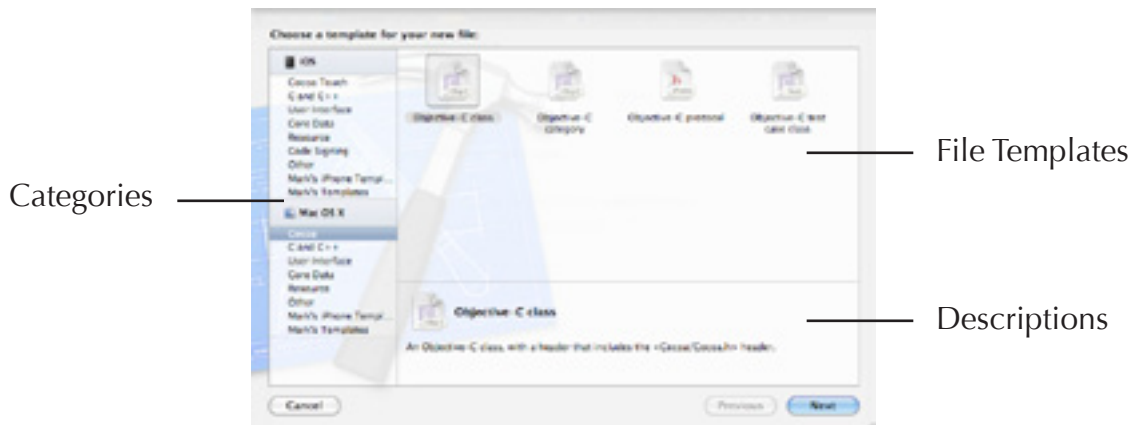- A notice saying who created the program and when.
- A copyright notice.

Figure 1.5

New File Assistant

Mac File Types

Xcode 4 has the following file template groups for Mac OS X programs:

- Cocoa
- C and C++
- User Interface
- Core Data
- Resource
- Other

Cocoa

The Cocoa group contains Objective-C source code files. There are five types of Cocoa files you can add to your project: Objective-C class, Objective-C category, Objective-C protocol, Objective-C test case class, and Objective-C class extension. A category adds methods to an existing class, which lets you add methods to Cocoa classes and lets you break up large classes into multiple files. A protocol is a header file that contains a list of methods to implement. Any class can implement the protocol. A test case class is used with unit testing, which tests the methods in a class to make sure they're correct.

If you create an Objective-C class, you must specify the superclass, the class from which your newly created class inherits. The combo box to choose a superclass has the following superclasses: `NSObject`, `NSDocument`, `NSView`, `NSViewController`, and `NSWindowController`. If you want a different superclass, enter it in the combo box's text field. Inherit from `NSObject` if you're not sure of what superclass to use.

When you create an Objective-C category, you get to specify the class on which you're creating the category. Enter the class in the Category on text field.

If you create an Objective-C test case, make sure you have a unit test bundle target in your project. Choose File > New > Target to add a unit test bundle target to your project. Make sure the test case gets added to the unit test bundle target, not your other targets.

Apple added the Objective-C class extension file template in Xcode 4.4. A class extension file is a header file that allows you to declare part of a class's interface outside of the class's public header file. The interface is the `@interface` section in Objective-C header files. The main use of a class extension is to place private variables you don't want other developers to see. Place the variables you want other developers to see in the class's public header file, and place the ones you don't want them to see in the class extension file.

C and C++

There are three C and C++ files you can add to your project: C file, C++ file, and header file. If you add a C++ file, Xcode creates a corresponding header file for you.

User Interface

Use the User Interface group to add a xib file to your project. There are five xib files to choose from for Mac applications: application, empty, main menu, view, and window. When you create a Cocoa application project, the project contains a xib file with a main menu so you shouldn't need to create an application xib or a main menu xib. You're most likely to create a view xib file or a window xib file.

Core Data

The Core Data group contains files for Core Data applications. The data model and mapping model files allow you to use Xcode's data modeling tools in Core Data applications. Refer to Chapter 5, "Modeling Tools", for more information on data models and mapping models.

In addition to the data model and mapping model files, the Core Data group has a `NSManagedObject` subclass. Core Data entities must inherit from `NSManagedObject` or a `NSManagedObject` subclass. Creating a `NSManagedObject` subclass lets you create a custom class for your entities to inherit from.

Resource

The Resource group has three files: property list, RTF File, and strings file. A property list is an XML file that stores persistent hierarchical application data. They are useful for storing small amounts of strings and numbers. A common case where you would use a property list is to store user preferences in your application.

An RTF file is a text file that can contain styled text.

As you can tell from its name, a strings file contains a list of strings. Strings files allow you to keep string data separate from your code. They are useful if your application supports multiple spoken languages like English, French, Spanish, and Japanese. Place your text in strings files to make your localization go more smoothly.

Other

The files in the Other group don't fit into the other groups. If you need an assembly language file, an empty file, or a shell script for your project, you'll find them in the Other group.

An exports file contains a list of exported symbols, such as class, function, and variable names. Xcode normally exports all the symbols from your project. Unless other applications are going to be calling the functions in your project, you can stick with Xcode's default behavior and not bother with exports files.

If you want to limit the symbols your project exports, add an exports file to your project and add the symbols you want to export to the exports file. Use the name of the exports file as the value of the Exported Symbols File build setting, which is part of the Linking collection. Framework and library projects are the projects most likely to use an exports file.

A configuration settings file is a text file that contains build settings. If you find yourself changing some build settings for each project, place those settings in a configuration settings file. I have more information about configuration settings files in the "Configuration Settings Files" section in Chapter 6, "Building Projects".

iOS File Types

Xcode has the following groups for iOS projects:

- Cocoa Touch
- C and C++
- User Interface
- Core Data
- Resource
- Other

Cocoa Touch

The Cocoa Touch group contains Objective-C source code files. You can create an Objective-C class, an Objective-C category, an Objective-C protocol, an Objective-C test case class, and an Objective-C class extension. A category adds methods to an existing class, which lets you add methods to Cocoa Touch classes and lets you break up large classes into multiple files. A protocol is a header file that contains a list of methods to implement. Any class can implement the protocol. A test case class is used with unit testing, which tests the methods in a class to make sure they're correct.

If you create an Objective-C class, you must specify the superclass, the class from which your newly created class inherits. The combo box to choose a superclass has the following superclasses: `NSObject`, `UIView`, `UIViewController`, `UITableViewCell`, and `UITableViewController`. If you want a different superclass, enter it in the combo box's text field. Inherit from `NSObject` if you're not sure of what superclass to use.

When you create a `UIViewController` or `UITableViewController` subclass, you have options to target the controller for iPad and add a xib file for the view controller's user interface.

When you create a `UIViewController` subclass, you have options to target the controller for iPad, create a `UITableViewController`, and add a xib file for the view controller's user interface. Use the Subclass of combo box to create a `UITableViewController`.

If you create an Objective-C category, you get to specify the class on which you're creating the category. Enter the class in the Category on text field.

If you create an Objective-C test case, make sure you create a unit test bundle target first. Choose File > New > Target to add a unit test bundle target to your project. Make sure the test case gets added to the unit test bundle target, not your other targets.

Apple added the Objective-C class extension file template in Xcode 4.4. A class extension file is a header file that allows you to declare part of a class's interface outside of the class's public header file. The interface is the `@interface` section in Objective-C header files. The main use of a class extension is to place private variables you don't want other developers to see. Place the variables you want other developers to see in the class's public header file, and place the ones you don't want them to see in the class extension file.

C and C++

There are three C and C++ files you can add to your project: C file, C++ file, and header file. If you choose to create a C++ file, Xcode creates a corresponding header file for you.

User Interface

Use the User Interface group to add a xib file or storyboard file to your project. There are four xib files to choose from for iOS applications: application, empty, view, and window. Since iOS application projects include a xib, you most likely will not need to create an application xib. View and window xib files are the ones you're most likely to create.

Choosing the storyboard file adds a blank storyboard file to the project. Read the section "Storyboarding" in Chapter 4, "Creating User Interfaces for iOS Applications", for more information on storyboarding.

When you select a file from the User Interface group and click the Next button, you should see a Device Family menu. This menu determines whether Xcode adds an iPhone xib or an iPad xib to your project.

NOTE

The xib files for iOS in the file template library are iPhone-sized. To create an iPad-sized xib file, choose File > New > File.

Core Data

The Core Data group contains files for Core Data applications. The data model and mapping model files allow you to use Xcode's data modeling tools in Core Data applications. Refer to Chapter 5, "Modeling Tools", for more information on data models and mapping models.

In addition to the data model and mapping model files, the Core Data group has a `NSManagedObject` subclass. Core Data entities must inherit from `NSManagedObject` or a `NSManagedObject` subclass. Creating a `NSManagedObject` subclass lets you create a custom class for your entities to inherit from.

Resource

The Resource group has the following files: settings bundle, property list, RTF File, strings file, GPX file, and GeoJSON file. A settings bundle specifies the application's settings.

A property list is an XML file that stores persistent hierarchical application data. They are useful for storing small amounts of strings and numbers. A common case where you would use a property list is to store user preferences in your application.

An RTF file is a text file that can contain styled text.

As you can tell from its name, a strings file contains a list of strings. Strings files allow you to keep string data separate from your code. They are useful if your application supports multiple spoken languages like English, French, Spanish, and Japanese. Place your text in strings files to make your localization go more smoothly.

GPX files specify a route or location to simulate. Use a GPX file to simulate running an application from a different city than your current location.

Apple added the GeoJSON file template in Xcode 4.5. Use a GeoJSON file to store geographic data when generating routes from one location to another.

Other

The files in the Other group don't fit into the other groups. If you need an assembly language file, an empty file, or a shell script for your project, you'll find them in the Other group.

The Other group also contains a resource rules file. Resource rules files are optional. They contain instructions for copying resources like audio and graphics files to the application bundle when building the application. You would need a resource rules file if you wanted to override the default method of copying resources to the application bundle.

The iOS Other group contains one additional file: configuration settings file. A configuration settings file is a text file that contains build settings. If you find yourself changing some build settings for each project, place those settings in a configuration settings file. I have more information about configuration settings files in the "Configuration Settings Files" section in Chapter 6, "Building Projects".

Fixing the Copyright Notice

In Xcode 4.4 Apple added an Organization Name text field to the New Project Assistant that specifies the name to appear in the copyright notice of the files you create. Adding the Organization Name text field reduces the need to fix the copyright notice, but if you created your project in an older version of Xcode, the material in this section may help you.

If you look at the copyright notice in the files you create, it may contain the text MyCompanyName. Unless you happen to work for a company called MyCompanyName, you want Xcode to use either your name or your company name in the copyright notice. To change the company name, open System Preferences and click the Users & Groups button. Select your account and click the Address Book Card Open button. Add a company name to your card.

When you use Address Book to change the company name, every subsequent file you create uses that company name. But you may want the files in a single project to have a different company name. You might be hired by another company to write a program for them, and you want that company's copyright to appear in the program you're writing.

To set the company name for a single project, select your project file in the project navigator. Open the file inspector by choosing View > Utilities > Show File Inspector. In the Project Document section of the file inspector is the Organization text field. Use the Organization text field to change the company name for this project. Any source code files you create for the project will place the new company name in the copyright notice.

Adding Files You've Already Created

Many applications contain more than just source code. They contain graphics, sound, and data files. When working with graphics, sound, and data files, you normally create them in another application and add them to your Xcode project so they get bundled with your application when Xcode builds it. You also might have existing source code files you want to add to a project. To add files you've already created to a project, choose File > Add Files to ProjectName. An Open panel opens that lets you find the files you want to add. After selecting files to add, click the Add button to add the files.

At the bottom of the Open panel are options for controlling how the files are added, which you can see in Figure 1.6. Selecting the Copy items into group's destination folder (if needed) checkbox tells Xcode to copy the files you added into your project's folder if the files aren't already there.

Adding a Folder of Files

If you're adding a folder of files, you have two options: recursively create groups for any added folders and create folder references for any added folders. When you recursively create groups, Xcode creates a group in the project navigator for each folder you add. Recursively create groups when you need to access the folder's files in Xcode. If you're adding a folder of source code files, you should recursively create the group because you want to be able to examine and edit the source code files.

When you create a folder reference, Xcode adds the folder to the project, but not the files in the folder. Create a folder reference when you don't need to access the folder's 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 editing 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.
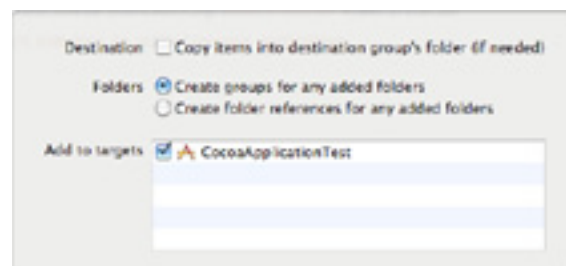

Figure 1.6

Options for adding existing files and folders to a project.

Adding Files to Targets

Below the adding folder options is a list of the targets in your project with a checkbox next to each target. If you select the checkbox for a target, the files you add will be added to that target. For an application target, files that you want in the application bundle should be added to the targets. Examples of files that should be added to the target are source code files, xib files, and image files. Header files, Xcode configuration settings files, and precompiled headers are examples of files that should not be added to the target.

Adding Frameworks and Libraries to a Project

If you're writing a program consisting mostly of GUI code, you can get away with using just the frameworks that are included when you create a Cocoa or Cocoa Touch application project. Many popular Apple technologies, such as OpenGL, GameKit, and Core Audio, require you to add a framework to your project to use the technology in your application.

Take the following steps to add a framework or library to your project:

1. Open the project editor by selecting the project file in the project navigator.
2. Select a target from the target list on the left side of the project editor.
3. Click the Build Phases button at the top of the editor.
4. Click the disclosure triangle next to the Link Binary with Libraries build phase.
5. Click the + button to add frameworks and libraries to the target.
6. A sheet opens with a list of frameworks and libraries. Select the frameworks you want to add, and click the Add button.

After adding frameworks to your project they will appear under the project in the project navigator. Drag the frameworks to the Frameworks folder if you want.

Source Trees

A source tree is a root path to place files that you want multiple people to work on. Source trees allow your project's files to remain independent of the project folder. The point of using source trees is to allow multiple people to work on a project. You can transfer the project to other people's computers without messing up the project's file references. To create a source tree, open Xcode's Locations preferences, click the Source Tree tab, and click the + button in the lower left corner of the preferences window.

There are three pieces of information you must supply to add a source tree.

- Setting Name is the name of the tree. Everyone who wants to use a source tree must give it the same setting name on their Macs.
- Display Name is the name Xcode shows for the source tree.
- Path is the location of the source tree on your hard disk.

Each person who wants to use a source tree can store it wherever they want on their hard disk. As long as everyone uses the same name for the source tree, everyone can use it. If you want other people to work on your projects, copy the files from the location of the source tree on your Mac to the location of the source tree on their Macs.

Removing Files from a Project

To remove a file from a project, select the file from the project navigator and press the Delete key. An alert opens asking if you want to delete the file. One of the buttons is Remove Reference Only. If you click this button Xcode removes the file from the project but does not delete it.

The second button depends on the version of Xcode you're running. Newer versions of Xcode have a Move To Trash button. Clicking this button removes the file from the project and moves it to the Trash.

Older versions of Xcode have a Delete button. Clicking the Delete button removes the file permanently from your hard disk. It does not move the file to the Trash. If you have a Delete button, I recommend clicking the Remove Reference Only button. Clicking the Remove Reference Only button and using the Finder to move the file to the Trash takes much less time than recovering a file you permanently deleted by mistake.

Renaming a Project

Naming your project is an important decision because Xcode uses the project name as the application name for application projects. If you want to change the name of your application, the easiest thing to do is rename the project.

Renaming a project is more involved than changing its filename in the Finder. Select the project in the project navigator and open its file inspector by choosing View > Utilities > Show File Inspector. At the top of the file inspector is the Project Name text field. Enter a new name and press the Return key. A sheet opens asking if you want to rename the project content items, which include the target name, the build product name, the precompiled

header file, and the `Info.plist` file. There are checkboxes to control what gets renamed. Click the Rename button to finish renaming the project. If you don't want to rename the project, click the Don't Rename button.

Modernizing a Project

When you open a project, Xcode evaluates it to make sure it conforms to the latest standards. Open the issue navigator to see if anything needs to be updated. If Xcode finds any outdated settings, they appear as a Project Modernization warning in the issue navigator. You can also choose Editor > Validate Settings to check for outdated settings.

Selecting the Project Modernization warning from the issue navigator or choosing Editor > Validate Settings opens a sheet with a list of modernization issues. Selecting the checkbox next to an issue tells Xcode to make the change when you click the Perform Changes button. Click the Don't Perform Changes button to keep the existing project settings.

What does Xcode check when it looks for modernization issues? Xcode checks for outdated build settings, file formats, and SDKs. Checking for outdated SDKs makes working with old Xcode projects easier in Xcode 4. If you have ever opened an old Xcode project and been unable to build it due to a missing SDK, you will appreciate Xcode's modernizing capabilities.

Workspaces

A workspace groups multiple projects in one window. Xcode treats the workspace as a single unit. All the projects in the workspace share the same build directory. Files in one project are visible to the other projects in the workspace. Use a workspace if you have a group of related projects. Suppose you have one project that builds a library that a second project needs. Place both projects in a workspace and Xcode will build the projects in the proper order.

When working with workspaces keep in mind that a workspace is a container for projects. The projects exist outside of the workspace. You can place a project in multiple workspaces. You can remove a project from a workspace without affecting the project or other workspaces.

Creating a Workspace

There are two ways to create a workspace. First, you can create a workspace out of an existing project by opening the project and choosing File > Save As Workspace. Second, choose File > New > Workspace to create an empty workspace. I recommend giving the workspace a different name than your project to avoid confusion.

When you create a workspace, you'll notice it looks a lot like Xcode's project window. The only visual difference between a workspace window and a project window is a workspace can contain multiple projects.

Adding Projects to a Workspace

There are three ways to add a project to a workspace. First, drag the project to the workspace. Second, choose File > Add Files to WorkspaceName. Third, right-click in the project navigator and choose Add Files to WorkspaceName.

When adding a project to a workspace, make sure you have not selected a project in the project navigator. If you select a project, the project you're trying to add to the workspace gets added to the selected project, not the workspace. The safest way to add a project to the workspace is the third way. Right-click in the project navigator below the projects and choose Add Files to WorkspaceName.

Organizer

The Organizer is an auxiliary window to perform tasks that aren't tied to a particular project. Use the Organizer to do things you can't do from the project window.

Opening the Organizer

Choose Window > Organizer or click the Organizer button in the project window's toolbar to open the Organizer. At the top of the Organizer are five buttons that correspond to the five sections of the Organizer: Devices, Repositories, Projects, Archives, and Documentation. The Devices section is for iOS developers. Skip to the section "Organizer for iOS Applications" to learn more about what you can do when you click the Devices button. If you're a member of Apple's paid Mac developer program, clicking the Devices button allows you to add your Mac to the paid developer program's portal.

The Repositories section shows your version control repositories. Use the repositories window to add new repositories and examine your commit messages. Read Chapter 8, "Version Control", for more information on using version control in Xcode.

The Projects section contains a list of recently opened projects. Use this section to open projects, remove projects from the Organizer, examine snapshots, and delete derived data, like build products and object files. Select a project from the list on the left side of the Organizer to access the project's snapshots and derived data. Right-click the selected project to open the project, reveal the project in the Finder, or remove the project from the Organizer.

The Archives section shows your archived applications. Archive each version of a shipping application so you can test that particular version for bugs that users report. The Archived Applications section lists the project builds you archived when choosing Product > Archive. An archived application is a compressed archive of an iOS or Mac application, similar to a zip archive. From this section you can validate your application to make sure it is ready to submit to the App Store, share copies of the application with testers, and submit the application to the App Store. When sharing a Mac application, you can share it as a Mac App Store package, an application package, or as an archive. You will be asked for a save location. When sharing an iOS application, you can share it as an App Store package or as an archive.

The Documentation section gives you a documentation window to read developer documentation. The section "Reading Developer Documentation" in Chapter 2, "Editing Source Code", contains detailed information on reading documentation.

Organizer for iOS Applications

When you click the Devices button at the top of the Organizer, you will see two sections on the left side of the Organizer: Library and Devices. The Library section has the items Provisioning Profiles, Software Images, Device Logs, and Screenshots. Older versions of Xcode may have a Developer Profile section. The Devices section has the items Provisioning Profiles, Applications, Console, Device Logs, and Screenshots. If you haven't connected a device to your Mac, you will not see the Provisioning Profiles, Applications, and Console items.

After reading the previous paragraph you noticed both the Library and Devices sections have Provisioning Profiles, Device Logs, and Screenshots items. The Devices section stores the provisioning profiles, device logs, and screenshots for a single device while the Library section stores all profiles, logs, and screenshots.

Developer Profile

If your version of Xcode has a Developer Profile section, it shows a list of your identities and provisioning profiles. The main use of the developer profile is to transfer your iOS developer information to a new Mac. Export your developer profile to a file, copy the file to the new Mac, and import the profile on the new Mac.

Newer versions of Xcode place this information in the Provisioning Profiles section.

Provisioning Profiles

Clicking the Provisioning Profiles item shows a list of your provisioning profiles. The Organizer displays the name of each profile and the expiration date. The provisioning profile tells the device to accept applications that you signed. To add a provisioning profile to your device, connect the device to your Mac and drag the provisioning profile to the device's icon in the Organizer.

If you select the Automatic Device Provisioning checkbox, Xcode handles the provisioning for you. Click the Refresh button. If you have a valid provisioning profile, clicking the Refresh does nothing. But if you don't have a valid provisioning profile, you'll be asked for your username and password. Xcode creates a team provisioning profile for you and copies it to your device. iOS provisioning profiles last for one year. Expired provisioning profiles are the most common cause of invalid provisioning profiles.

Software Images

When you upgrade your device to the latest version of iOS, a software image for the latest version is copied to your Mac's hard drive. Selecting Software Images lets you view the images that have been copied to the hard drive and delete them if you don't want them on your hard drive.

Device Logs

Clicking the Device Logs item shows a list of crash logs and other logs for your iOS devices. The list tells you the application that created the log, what caused the log entry (examples include crashes and low memory conditions), and the date it occurred. Selecting a log from the list displays it in the Organizer.

If your iOS application crashes, drag the crash log to the Device Logs section of the Organizer. Xcode will add symbols to the crash log to make debugging the crash easier.

In Xcode 4.5 and later the bottom of the Organizer has buttons to import, export, and resymbolicate crash logs. Select a crash log from the list to export or resymbolicate the log.

Screenshots

The Screenshots section shows screenshots of your iOS applications. The Organizer has two panes for screenshots. The left pane has a list of all the screenshots you've taken. Selecting a screenshot shows a full-size version of it in the right pane. Clicking the Save as Launch Image button makes that screenshot the launch image for a project. When you click the Save as Launch Image button, a window opens with a sheet of open projects. Select a project and click the OK button. When you click the OK button, Xcode adds the screenshot to the project.

To take a screenshot of your device, select Screenshots under Devices. Click the New Screenshot button in the lower right corner of the organizer. The screenshot appears on the right side. To remove a screenshot, select it from the screenshot list and click the Delete button at the bottom of the Organizer.

Clicking the Export button saves the screenshot as a PNG file. Selecting multiple screenshots and selecting the Compare checkbox lets you view the differences between the screenshots. Use the Tolerance slider to adjust the colors of the differences.

Devices

When you connect a device to your Mac, the device appears in the Devices section of the Organizer and a dot appears next to the device. If the color of the dot is not green, it means the device is connected, but not available for development. Select the device and click the Use for Development button. When you tell Xcode to use the device for development, a green dot appears next to the device.

Selecting the device provides general information about the device, including the iOS version, the provisioning profiles, the installed applications, the number of crash logs, and the number of screenshots. Clicking the focus button next to the provisioning profiles, applications, device logs, or screenshots takes you to that section.

After you set the device for development, you can use the Organizer to add provisioning profiles to the device, add applications to the device, view the console, view crash logs, and take screenshots of the device. To be able to add applications and provisioning profiles to your device, you must join the iOS Developer Program and pay the annual membership fee.

The Devices section has the following entries:

- Provisioning Profiles
- Applications
- Console
- Device Logs
- Screenshots

You must connect a device to your Mac to see the Provisioning Profiles, Applications, and Console entries. The Applications entry shows the applications installed on the device. Use this section to add applications to and remove applications from the device.

The Console entry displays the console log for the device. Click the Save Log As button at the bottom of the Organizer to save the log on your Mac.

The Provisioning Profiles, Device Logs, and Screenshots entries in the Devices section store the provisioning profiles, device logs, and screenshots for a single device. Read the "Provisioning Profiles", "Device Logs", and "Screenshots" sections earlier in this chapter for more information on provisioning profiles, device logs, and screenshots.