

Chapter 1

Xcode Projects

In this chapter, you'll learn how to create a project, add files to the project, and create your own project and file templates. You'll also learn about Xcode's project window and Organizer 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 Interface Builder 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.

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 3.2 has the following project categories:

- Application
- Framework and Library
- Application Plug-In
- System Plug-In
- Other
- iPhone Application
- iPhone Library
- Java

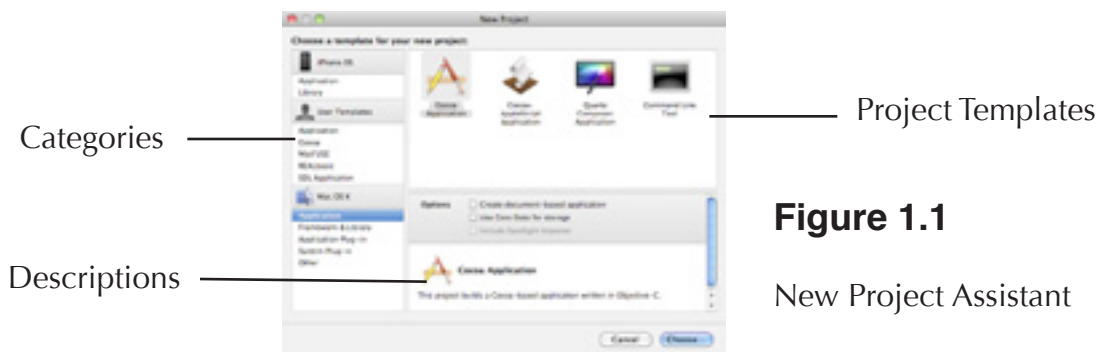


Figure 1.1

New Project Assistant

2 Chapter 1: Xcode Projects

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 in the bottom of the Project Assistant.

Some projects require additional information. If you're writing a command-line application, you must tell Xcode what language you're using: C, C++, or Objective-C.

After choosing the type of project you want to make, click the Next button. Tell Xcode the name of your project and where you want to store it, then click the Finish button. Congratulations! You've created an Xcode project.

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`, `ProjectNameAppDelegate.h`, and `ProjectNameAppDelegate.m`, where `ProjectName` is the name of your project.
- The Cocoa, AppKit, Foundation, and Core Data frameworks. The AppKit framework contains the user interface portions of Cocoa, and the Foundation framework contains the base classes from which the AppKit classes inherit. I have no idea why the Core Data framework is part of a Cocoa application that doesn't use Core Data.
- A xib file, `MainMenu.xib`, containing the user interface. Use Interface Builder to modify the interface.
- Two property list files, `Info.plist` and `InfoPlist.strings`. 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 configuration information for the application. The `InfoPlist.strings` file contains localized configuration information. There will be one `InfoPlist.strings` file for each spoken language your application supports.
- A prefix header, `ProjectName.pch`, where `ProjectName` is the name of your project. Xcode precompiles the header files in the prefix file, which makes your project compile faster.

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, AppKit, and Foundation frameworks. It also will include a data model file with the extension `.xcdatamodel`. 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 chose 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 choosing `Build > Build and Run`. 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 three Cocoa projects: Cocoa Application, Cocoa AppleScript Application, and Quartz Composer 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.

The Quartz Composer application project is similar to the Cocoa Application project. The main difference is the Quartz Composer application includes a Quartz Composer file; it has the extension `.qtz`. Quartz Composer comes with the Xcode Tools and lets you create graphics compositions without having to write code. If your Cocoa application uses Quartz Composer compositions, create a Quartz Composer application project.

Document vs. Non-document Applications

For Cocoa applications you have 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.

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.

4 Chapter 1: Xcode Projects

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 command line (launch the Terminal application to get access to the command line) 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 Services and Core Foundation frameworks. 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 Frameworks and libraries 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 four categories of projects:

- Libraries
- Frameworks
- Bundles
- JNI Library

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

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 REALbasic.

Bundles can be created using the Cocoa and Core Foundation frameworks.

JNI Library

Java Native Interface (JNI) library projects let you access code written in other programming languages, such as C, C++, or Objective-C. If you wrote some non-Java code you wanted to use in a Java program, use a JNI project.

Application Plug-In Projects

There are project templates to create plug-ins for Automator, Address Book, Installer, WebKit, Quartz Composer, and Interface Builder. The WebKit plug-in creates plug-ins for applications that use WebKit, such as Safari and Dashboard. Use the Interface Builder plug-in to create custom controls and custom views.

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

These projects are lower level than the application plug-in projects.

- Audio Unit Effect
- Audio Unit Instrument
- Generic C++ Plug-In
- Generic Kernel Extension
- Image Unit Plug-in
- IOKit Driver
- Preference Pane (System Prefs)
- Quick Look Plug-In
- Screen Saver
- Spotlight Plug-In
- Sync Schema

The Audio Unit Effect and Audio Unit Instrument projects create plug-ins for Core Audio, Apple's low-level audio framework.

The Generic Kernel Extension 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 had a custom document type, a Quick Look plug-in would let the user examine a document by choosing File > Quick Look in the Finder.

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 he or she could synchronize their data with a laptop, cell phone, or iPod.

Other Projects

There are two templates in the Other group: an empty project and an external build system 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 program. 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++, Java, 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 Perl, Python, and Ruby so you don't have to install anything to use those languages.

iPhone Application Projects

These projects are used to write applications that run on the iPhone and iPod Touch. To be able to use iPhone application projects, you must install the iPhone SDK. There are six iPhone application templates.

- Navigation-Based Application
- OpenGL ES Application
- Tab Bar Application
- Utility Application
- View-Based Application
- Window-Based Application

If you're not sure what project template to choose, go with the window-based application or view-based application templates. They provide a starting point for any iPhone application. If you find out later that you want a tab bar or navigation bar in your application, you can add the tab bar or navigation bar later.

The navigation-based application and tab bar application templates provide a second `xib` file and a view controller class. If you're creating an application that uses a navigation bar or tab bar, the corresponding templates make things easier for you.

The OpenGL ES application template includes the OpenGL ES framework, which is a version of OpenGL for mobile devices, such as cell phones. If you're writing a game or an application that requires high-performance graphics, use the OpenGL ES application template.

8 Chapter 1: Xcode Projects

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.

If you're running the iPhone 3.0 SDK or later, the navigation-based application and window-based application projects give you the option of using Core Data for storage.

iPhone Library Projects

The iPhone 3.0 SDK added a library project template. The template lets you create a static library using the Foundation framework.

Java Projects

Java projects do not appear in the New Project Assistant when you create a new project. To create a Java project, you must open the Organizer window by choosing Window > Organizer. Click the + button in the lower left corner and choose New From Template > Java Templates. There are five templates to choose from: Applet, Application, Signed Applet, Tool, and Web Start application. For more information on the Organizer, read the "Organizer" section later in this chapter.

Applets are programs that will run in another application. Many websites use Java applets that run in your web browser. Signed applets are applets that contain a digital signature.

Application projects build GUI applications. Web Start is a technology that lets users easily download and launch Java applications from the Internet.

Tool projects create command-line applications. If you're learning Java programming, you should create Tool projects.

Creating Project Templates

Xcode has a lot of project types to choose from, but you may need a project type that Xcode doesn't supply. If you write a lot of OpenGL programs, you would like to have an OpenGL application project that includes the OpenGL framework so you don't have to add the framework every time you create a project. When Xcode's project types don't fit your needs, create a project template.

A project template is just an Xcode project. When you create a new Xcode project using your project template, Xcode creates a project that contains everything in the project template. There are four tasks to perform to create a project template.

- Duplicate one of Apple's project templates.
- Modify the project in Xcode.
- Move the template to the user templates location.
- Change the project's description.

Duplicating Apple's Project Templates

The easiest way to create a project template is to duplicate one of Apple's templates and modify it to suit your needs. You can find Apple's iPhone project templates in the following location:

```
Developer/Platforms/iPhoneOS.platform/Developer/Library/  
Xcode/Project Templates
```

The rest of Apple's project templates are in the following location:

```
Developer/Library/Xcode/Project Templates
```

Select the project type that most closely matches the template you want to create. Make a copy of that project folder. This copy is your template folder.

The project folder's name is what appears in the New Project Assistant. Change the name of your template folder.

Modifying the Template in Xcode

After duplicating one of Apple's project templates, you must modify it to suit your needs. Otherwise there would be no reason to create a project template.

Open the project file that resides in your template folder. Make the changes you need to make to the project. Add frameworks and files to the project. Remove frameworks and files you don't need. Add code to a source code file. Do whatever you have to do to make the project a suitable template.

Moving the Template

After creating the project template of your dreams, you must move the template folder for the project. You don't want your project templates in the same folder as Apple's templates. If you install a newer version of Xcode at a later date, your templates will be erased.

10 Chapter 1: Xcode Projects

Where should you move the template folder? I recommend placing it in the following folder:

```
/Library/Application Support/Developer/Shared/Xcode/Project  
Templates/GroupName
```

Where `GroupName` is the name you want to appear under User Templates on the left side of the New Project Assistant. Name your group something like My Templates. If you don't supply a `GroupName` folder, your template won't be visible in the New Project Assistant.

You may have to manually create the last five folders in the path I recommended. If you use my recommended path to store your project template, the templates will be accessible to all user accounts on your Mac and to all versions of Xcode you have installed on your Mac. Making your project templates accessible to all users and Xcode versions shouldn't be a problem in most cases, but suppose you wanted to restrict access of one of your project templates to your user account and Xcode 3.2. You would place your project template folder in the following location:

```
/Users/YourUsername/Library/Application Support/  
Developer/3.2/Xcode/Project Templates/GroupName
```

Changing the Project's Description

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

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

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

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.

To rename a project, choose Project > Rename. A sheet opens, which you can see in Figure 1.2. Use the text field to rename the project. Select the Take snapshot before renaming checkbox if you want to take a snapshot of the project before renaming. Click the Rename button to finish renaming the project.

Below the text field is a list of files to change. You can rename the project file, targets, build products (application, library, or framework name), precompiled headers, property list files, and any other files that were based on the old project name. Xcode is initially set up to rename all the files it can. If you want to keep the old name for some files, deselect the appropriate checkboxes.

Project Window

The project window, shown in Figure 1.3, is your project's home when working in Xcode. It has the following components:

- Toolbar
- Favorites bar
- Groups and Files list
- Detail view
- Status bar
- Editor

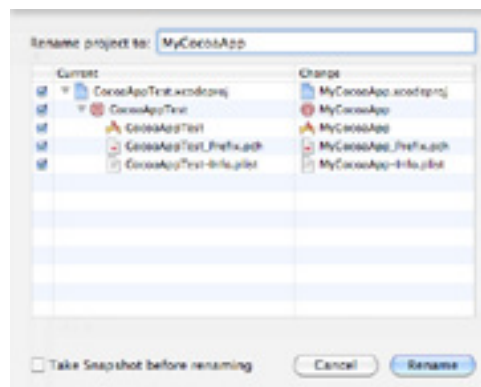


Figure 1.2

Rename project window

12 Chapter 1: Xcode Projects

I'm going to cover the toolbar and the Groups and Files list shortly. The favorites bar is initially invisible. Choose View > Layout > Show Favorites Bar to make the favorites bar visible. Use the favorites bar to store items you want to quickly access, such as source code files. To add an item to the favorites bar, select the item and drag it to the favorites bar.

The detail view works with the Groups and Files list. When you select a group from the Groups and Files list, the detail view displays all the files in the group.

The status bar reveals the progress of lengthy tasks, such as building your project. The editor is initially invisible. Drag the splitter bar at the bottom of the window to show the editor. Choosing View > Zoom Editor In replaces the detail view with the editor. If you prefer to edit your source code in a separate window, leave the editor in the project window invisible. The point of showing the editor in the project window is to reduce window clutter by having only one window open.

Toolbar

At the top of the project window is the toolbar, which provides you easy access to tasks you perform most. Xcode places the following items on the toolbar by default:

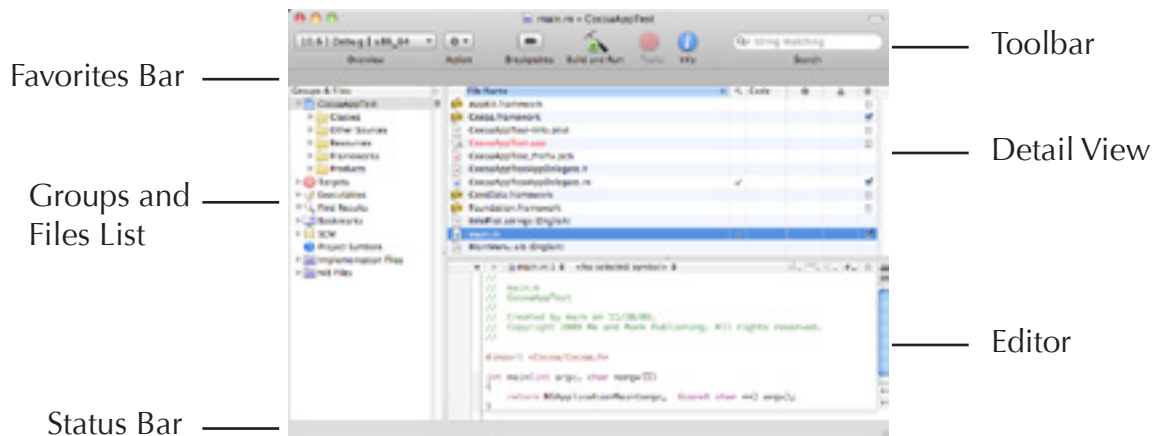


Figure 1.3

Project window

- The Overview menu lets you set the active SDK (the SDK Xcode will use to build your project), the active configuration (the build configuration used to build the project), the active target (the target Xcode creates when building your project), the active executable, and the active build architecture.
- The Action menu is a contextual menu that contains frequently used commands. What appears in the menu depends on what you select in the Groups and Files list. You can also open the Action menu by right-clicking an item in the Groups and Files list.
- The Breakpoints button determines whether you want to run or debug your project after building it. Clicking the Breakpoints button changes the Build button from Build and Run to Build and Debug and vice versa.
- The Build button builds your project and runs it, either in Xcode or in the debugger.
- The Tasks button will stop any programs you have executing in Xcode.
- The Info button opens the selected item's inspector. You will use this button to tweak settings for the project and the files in the project.
- The Search field lets you filter listings in the detail view. If you wanted to see only header files in the detail view, you would type `.h` in the search field.

If those items aren't the ones you use most, you can customize the toolbar to suit your needs. To customize the toolbar, choose View > Customize Toolbar or right-click the top of the window and choose Customize Toolbar. A window like Figure 1.4 will open. To add an icon to the toolbar, drag it to the toolbar. To remove an item from the toolbar, drag it off the toolbar. To rearrange items in the toolbar, drag the icon where you want it to appear.

You can also choose how the items on the toolbar appear. You can show the icon only, text only, or an icon with text. You can also make the icons smaller by selecting the Use Small Size checkbox. Click the Done button when you're finished tinkering with the toolbar.

NOTE

You can customize the toolbar for any Xcode window that has one. The steps you take are the same as customizing the project window toolbar.



Figure 1.4

Customize toolbar window

Groups and Files List

The Groups and Files list shows the contents of your project. Xcode projects have many files so Xcode places the files in groups. Right-clicking (Control-click if you have a one-button mouse) an item in the Groups and Files list opens a contextual menu. The contents of the contextual menu depend on the item you click, but the tasks you can perform from the contextual menu include adding files to a project, adding targets to a project, adding build phases to a target, compiling a single file, and building a target.

Project Name

The first entry in the Groups and Files list is the name of your project. If you select it, the detail view displays all the files associated with your project: source files, frameworks, property lists, and executable files. Figure 1.5 shows what the detail view looks like when you select the project name from the Groups and Files list.

The detail view displays the following information about each file in your project:

- An icon representing the type of file, such as framework, executable file, header file, or property list.
- The file's name.
- Role. You will not see a Role column unless you select a target from the Groups and Files list. Only frameworks and libraries have anything in the Role column. The Role column lets you determine if the framework is required to be linked or if it can be weak-linked.
- Does the file need to be built? If the file has a check in this column, it needs to be compiled.
- For source code files the Code column tells you the amount of code in the file. The Code column will be blank until you compile your project.
- The Errors column tells you how many compiler errors appeared in this file. This column applies only to source code files. You want the Errors column to be blank, which means your source code files have no errors.

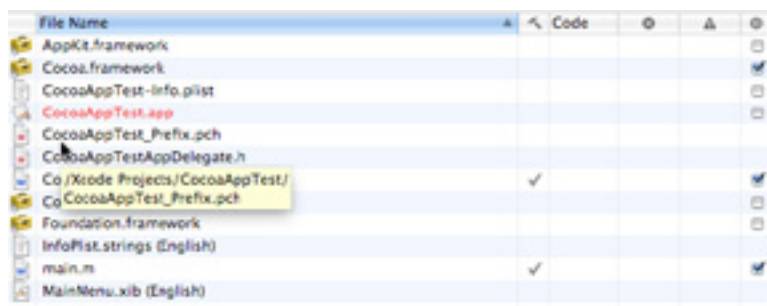


Figure 1.5

Project window detail view

- The Warnings column displays the number of compiler warnings for this file. You want this column to be blank as well as the Errors column.
- The Target column has a checkbox for your source code files and frameworks. If the checkbox is selected, the file is part of the active target.

Weak-linking deserves some additional explanation. When should you weak-link a framework? Weak-link a framework if it was introduced in a later version of Mac OS X than a version of Mac OS X you want to support. Suppose you're writing an application that runs on Mac OS X 10.5 and 10.6, and you want to take advantage of the OpenCL framework. The OpenCL framework was introduced in Mac OS X 10.6. By weak-linking the OpenCL framework, you can use the framework on 10.6, and your application will still run on 10.5.

Double-clicking a source code file or a property list file opens the file in a separate window. Double-clicking a xib file opens it in Interface Builder. Double-clicking a framework opens a window for it in the Finder. If you don't want to open separate windows, drag the splitter bar at the bottom of the project window to show the editor. Selecting a source code file or property list file from the project window will open the file in the editor pane. Xcode doesn't make the project windows very large. You will want to resize it if you're going to edit source code in the project window.

Xcode sorts the files alphabetically by default. By clicking one of the column headings, Xcode will sort the files by the column you clicked.

If you have a large project, you may have a hard time finding the files you're interested in. By clicking the disclosure triangle next to the project name in the Groups and Files list, Xcode displays folders for each group in your project. The initial group of folders a project has depends on the project you make, and you can add your own groups by choosing File > New Group. Selecting a folder will make that folder's files appear in the detail view.

Targets

As the name implies, the Targets section lists your project's targets. What is a target? A *target* is a set of instructions to build a final product from the files in your project. Examples of final products are applications, libraries, and frameworks. When you create an application project, Xcode automatically creates a target for building an executable file.

Click the disclosure triangle next to the Targets entry to see your project's targets. Selecting a target fills the detail view with the files and frameworks that are part of the target. Xcode projects initially have one target. The section "Adding Targets" in Chapter 4, "Building Projects", explains how to add targets to a project. If you have multiple targets in your project, use the Overview pop-up menu to choose the active target. The section "Working with Targets" in Chapter 4, "Building Projects", provides more information on targets.

Executables

The Executables group lets you tinker with the settings for the executable files your project creates. Obviously this group is important only for projects that create executable files, like application projects. Command-line applications are the ones most likely to require tweaking the settings of an executable file.

Selecting Executables makes all the executable files for your project appear in the project window. Most of the time you will have only one executable file, but if you add targets to your project, there's the possibility of multiple executable files. Selecting an executable file in the project window and clicking the Info button opens the executable file's inspector. The panel has four tabs.

- General
- Arguments
- Debugging
- Comments

Click the Comments tab to add notes about the executable file's settings.

General Panel

The general panel, shown in Figure 1.6, is where you set the following information about the executable file:

- The path to the executable file.
- The type of frameworks to load when running your program.
- The program to use for standard input and output.
- The working directory.

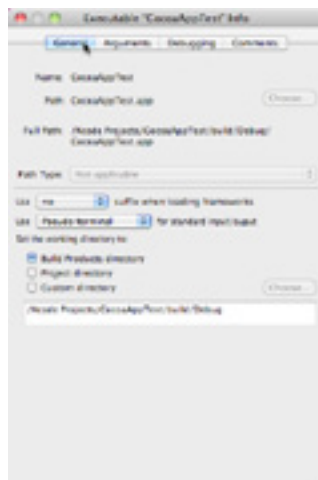


Figure 1.6

Executable inspector

For bundled applications the path is ExecutableName.app. Most Mac OS X applications are bundled. They appear in the Finder as a single executable file, but beneath the surface lays a web of directories. Inside this web is the actual executable file. Because bundles are complex to navigate manually, Xcode does not let you modify the path for bundled applications. If your application is not bundled, you can click the Choose button to set a path.

Use the Use suffix when loading frameworks pop-up menu to tell Xcode what type of framework to load. Xcode can load one of three versions of a framework.

- The standard variant is the default option and it runs the normal runtime library. This variant will be what you want most of the time.
- The debug variant provides additional debugging information while your program runs.
- The profile variant provides additional information for profiling your application to see where it spends the most time.

If you're writing a program without a GUI, you will appreciate the pop-up menu that lets you choose the program to use to display standard input and output. The program options are:

- Pseudo terminal, which is the default option. When you use pseudo terminal, command-line programs run in Xcode's console the way they would run from the command line.
- System console. When you use system console your program's output appears in the system log instead of Xcode's debugger console. The advantage of using the system console is your program's output is saved to a log file for you to look at later. I recommend not using system console if your program uses standard input to get information from the user.
- Pipe is the option to use if you're going to run your program from another computer. Pipe lets you transfer input and output between the two computers.

Last comes the radio button group to set your program's working directory, the initial directory where your program looks for files. Most Mac OS X applications do not deal with working directories; they use Cocoa's `NSBundle` class or Core Foundation's bundle functions to retrieve files from the application bundle. If you decide to set the working directory, the initial location is your product directory, which is the build folder in your project directory. You can choose to go with the project folder, or you can select an entirely different directory as your working directory.

Arguments Panel

Use the arguments panel to set the runtime arguments and environment variables your program uses. Command-line projects are the most likely to use runtime arguments. To add a launch argument, click the + button and enter the argument name. To temporarily disable an argument, deselect the checkbox next to the argument. To remove an argument, select it from the list and click the minus button.

An environment variable is an operating system global variable that applications can use to send data to and receive data from other applications. Environment variables allow programs to reconfigure themselves while they're running. Unix programs use environment variables more than Mac programs.

To add an environment variable, click the + button. Type in a variable name and give it a value. To temporarily disable an environment variable, deselect the checkbox next to the variable. To remove an environment variable, select it and click the minus button.

If you want to experiment with environment variables, add one by clicking the + button. Give the variable the name `MallocScribbling` and the value 1. Now when you run your program, the operating system will take the memory your program frees and fill the freed memory with garbage values. Normally the memory would remain unchanged when you free it. Setting the `MallocScribbling` environment variable lets you discover if you're accessing memory you've already freed; your program will crash if you access freed memory.

Debugging Panel

The debugging panel is where you set options related to debugging the executable file. I cover the debugging panel in more detail in Chapter 5, "Debugging with Xcode".

Find Results

The Find Results group lists the results of searches you made for this project. It helps when you search for a particular term in your project and want to look for it again later. Xcode stores the results temporarily; when you close your project, you lose the search results.

Bookmarks

The Bookmarks group lists all the bookmarks you've set. Bookmarks let you quickly reach a line in one of your source code files. To add a bookmark, move the cursor to a line in one of your files and choose Edit > Add to Bookmarks. Xcode opens a dialog box to name the bookmark. By default Xcode lists the source code file and line number, but you can give the bookmark any name you want. If you click the Bookmarks entry in the Groups and Files list, you will see the bookmark you created.

SCM

The software configuration management (SCM) group works with version control, which tracks the changes you make to files. Version control works by storing all your files in a repository. You check files out of the repository, make the changes you want to make to them, and check the files back in. The SCM group lists the files you have modified but haven't checked back in. If you're new to Xcode, the SCM group will be blank because you haven't set up version control yet. Version control is such a large subject, I devote an entire chapter to it. Refer to Chapter 6, "Version Control" for more information on using version control with Xcode.

Project Symbols

Selecting Project Symbols from the Groups and Files list provides lots of information. The project symbols group shows every class you wrote, the data members of the classes, every function you wrote, every constant you defined, every enumerated data type you created, and every macro you defined.

The project symbols listing has three columns. The first column is the name of the symbol. There's a symbol for every class, data member, function, constant, data type, and macro in your program. The second column describes the type of symbol. Table 1.1 provides a list of common symbols. The final column tells you the file and the line number where the symbol appears. Double-clicking a symbol opens up a new editor window and takes you to the symbol.

Table 1.1 Common Project Symbol Types

Symbol Type	Description
Class	A class name. You will have one of these symbols for each class you create in an object-oriented language like C++, Java, or Objective-C.
Class Method	Static functions in a class. Normally when you declare objects of a particular class, each object has an instance of each member function in the class. Static functions have one instance for the class. All objects of the class share the one instance of the static function.
Constant	If you use enumerated data types, each value in the data type has a Constant symbol.
Enum	The name of an enumerated data type.
Function	Functions that are not member functions of a class. C functions, C++ constructors, and C++ destructors have Function symbols.
Instance Method	Non-static member functions of a class. If you use an object-oriented programming language, most of the functions you write will have Instance Method symbols.
Instance Variable	The data members of a class.
Macro	Macros that you create with the <code>#define</code> statement. C programs use macros to define constants.
Type	If you use the <code>typedef</code> statement in C to define new data types, the new data types appear in the symbol list as Type symbols.
Variable	Variables declared outside of a class. If you use constants for special values to avoid hard-coding numbers, the constants appear in the symbol list as Variable symbols.
Modeled Class	A class that is part of a xib file or Xcode data model.
Modeled Method	A method that is part of a xib file or Xcode data model. Interface Builder actions are the most common modeled methods.
Modeled Property	A variable that is part of a xib file or Xcode data model. Interface Builder outlets are the most common modeled properties.

Smart Groups

Smart groups do not have the name Smart Groups. They're folders that group files using rules you specify. Xcode projects come with two smart groups: Implementation Files and NIB Files. The Implementation Files folder contains the AppleScript, C, C++, Java, Objective-C, Objective-C++, and shell script files in your project. The NIB Files folder contains your project's nib files, files with the extension `.xib` or `.nib`.

If your project doesn't contain many files, you won't need smart groups, but for large projects, creating your own smart groups makes locating files easier. To create a smart group, choose Project > New Smart Group. There are two types of smart groups: simple filter and simple regular expression. It doesn't matter which one you choose right now. You can change the type of smart group later.

The smart group you create initially has the name Simple Filter Smart Group or Simple Regular Expression Smart Group. You're going to want to change the name. Select the smart group you created from the project window and click the Info button. The group's inspector opens, as you can see in Figure 1.7.

From the inspector you can choose the name and folder icon for the group. You can also choose the starting point for grouping the files. By default it's the project folder, which means Xcode looks at all files in the project. You can limit the search to a specific folder inside the project if you wish.

In the Using Pattern text field, you type in the pattern Xcode uses to determine what files appear in the group. There's also a radio button determining whether you want to use a wildcard pattern or regular expression. That's why I said it didn't matter which type of smart group you created initially; you can change it here.

Wildcard patterns are easier to use. Type in the filter you want to use, and you're done. All the files that match the filter will appear in the group. If you wanted to create a smart group for header files, you would type in the pattern `*.h`. Every file with the extension `.h`, the extension that header files use, will appear in the group.



Figure 1.7

Smart group inspector

Regular expressions give you more power. Enter any Unix expression, and Xcode places files that satisfy the expression in the group. To see an example of a smart group that uses regular expressions, select the Implementation Files group and click the Info button. The Implementation Files group uses the following expression:

```
\.(c|cpp|C|CPP|m|mm|java|sh|sct)$
```

This expression says that any file that contains the characters `.c`, `.cpp`, `.C`, `.CPP`, `.m`, `.mm`, `.java`, `.sh`, and `.sct` appears in the group. These characters happen to be the default file extensions for C, C++, Objective-C, Objective-C++, Java, shell script, and AppleScript files.

Use the Save For pop-up menu to determine whether you want the group you created to appear in this project only or in all Xcode projects you create.

Breakpoints

The Breakpoints section lists all the breakpoints you have set in your program. Breakpoints tell the debugger to pause your program when it reaches a certain line of code or reaches a certain function. Breakpoints are essential to debugging your program. I have more information on breakpoints in the debugging chapter.

Project Window Layouts

The project window layout you get initially when creating a project is the default layout, but there are two other layouts to choose from. To change the project window layout, you must close all your projects. Open Xcode's preferences window and click the General button on the preference window's toolbar. Choose a layout from the layout menu.

All-In-One Layout

The all-in-one layout looks similar to the default layout. The major difference is the all-in-one layout is designed to have just one window open per project. The all-in-one layout has extra controls to keep multiple windows from opening. The all-in-one layout has tabs for the detail, project find, SCM results, and build windows so you can get to those windows without having to open separate windows, which is what occurs in the default layout. The toolbar in the all-in-one layout has a Page control on the left side that lets you switch between the project window and the debugger. The main advantage of the all-in-one layout is reduced window clutter.

Condensed Layout

The condensed layout, shown in Figure 1.8, has a simpler project window. Longtime Mac developers may notice the condensed layout looks similar to CodeWarrior's project window layout. The main differences between the condensed and default layouts is there is no detail view in the condensed layout and the Groups and Files list is split into three sections in the condensed layout: Files, Target, and Other. The main advantage of the condensed layout is a smaller project window that gets out of your way so you can concentrate on source code.

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 would add to a project include source code files, xib files, data files, graphics files, and audio files. You can create new files or add existing files to your project.

Adding New Files to the Project

The most common case of adding files to a project is creating new source code files. To create a new file, choose File > New File or use the Action menu in the project window and choose Add > New File. Creating a new file consists of two steps.

- Choosing a file type.
- Naming the file.

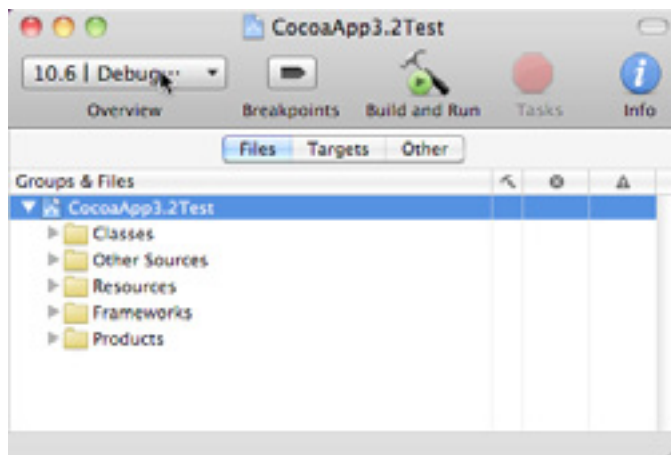


Figure 1.8

Condensed project window layout

Choosing a File Type

When you create a new file by choosing File > New File, the New File Assistant dialog box opens. Figure 1.9 shows what the dialog box looks like in Xcode 3.2. The left side of the dialog box contains a list of template groups. Selecting a group fills the dialog with the group'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.

The File Types

Xcode 3.2 has the following file template groups for Mac OS X:

- Cocoa Class
- C and C++
- User Interface
- Resource
- Interface Builder Kit
- Other

Cocoa Class

There are four Cocoa class files: Objective-C class, Objective-C protocol, Objective-C test case class, and AppleScript class. 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.

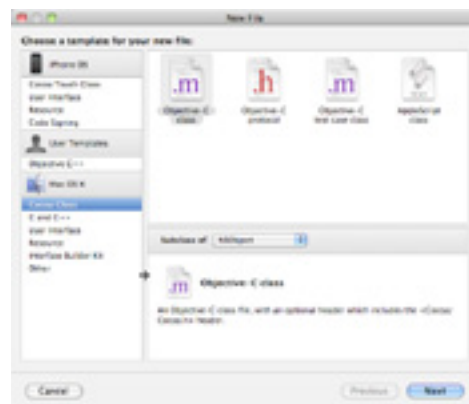


Figure 1.9

New File Assistant

If you choose Objective-C class or AppleScript class, you get to specify the superclass, the class from which your newly created class inherits. Both Objective-C and AppleScript classes can inherit from `NSObject`, `NSDocument`, and `NSView`. Objective-C classes can also inherit from `NSWindowController`. Inherit from `NSObject` if you're not sure of what superclass to use.

If you're going to create an Objective-C test case, make sure you create a unit test bundle target first. Choose Project > 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.

C and C++

There are three C and C++ files: C file, C++ file, and header file. If you choose to create a C file or a C++ file, Xcode will create a corresponding header file for you.

User Interface

Use the User Interface group to add a xib file to your project. There are five choices: 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 or a window xib.

Resource

The Resource group contains files that support your source code files. The Data Model and Mapping Model files allow you to use Xcode's data modeling tools in Core Data applications. Refer to the Modeling Tools chapter for more information on data models and mapping models.

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.

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 string files to make your localization go more smoothly.

A sync client description is a property list that registers clients with Mac OS X's Sync Server. Use a sync client description if you want to sync your application's data with other applications or with other devices, such as another Mac, an iPod, or a cell phone.

Interface Builder Kit

Interface Builder plug-in projects make use of the files in the Interface Builder Kit group. If you're not writing an Interface Builder plug-in, you can skip this section.

The class description is a text file that defines the class, its superclass, its actions, and its outlets. The Inspector and Inspector XIB files allow you to create an inspector for your custom control in Interface Builder. The inspector file contains the code and the Inspector XIB contains the user interface. The Interface Builder Integration file lets you customize the behavior of your view in Interface Builder. The Interface Builder Library NIB lets you create a component for Interface Builder's Library window.

Other

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

Add a Class Model file to your project if you want to use Xcode's modeling tools to examine your project's classes. I have more information on class models in Chapter 3, "Modeling Tools".

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 export 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 Chapter 4, "Building Projects".

iPhone File Types

If you install the iPhone SDK, you will have the following additional groups:

- Cocoa Touch Class
- User Interface
- Resource
- Code Signing

Cocoa Touch Classes

The Cocoa Touch Classes group contains source code files. You can create an Objective-C class, an Objective-C test case class, and a `UIViewController` subclass. If you choose to create an Objective-C class, you must choose the class your newly created class inherits from: `NSObject`, `UITableViewCell`, `UITableViewController`, and `UIView`. If you're not sure, subclass `NSObject`.

If you're going to create an Objective-C test case, make sure you create a unit test bundle target first. Choose Project > 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.

User Interface

The User Interface group contains xib files. There are four xib files to choose from: application, empty, view, and window. Since iPhone 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.

Resource Files

There are three files in the Resource Files group: data model, mapping model, settings bundle. The data and mapping models are for Core Data. Refer to Chapter 3 "Modeling Tools" for more information on data models and mapping models. The settings bundle specifies the application's settings.

Code Signing

The Code Signing group contains two files: entitlements file and resource rules file. Entitlements files are used for ad-hoc distribution. If you're beta-testing an application, use ad-hoc distribution to get your application to the testers.

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.

Naming the File

After selecting the type of file you want to create, click the Next button, which will take you to the second 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 created a C, C++, or Objective-C file, there will be a checkbox with the caption Also create `FileName.h`, where `FileName` is the name of your file. C, C++, and Objective-C programs normally have one header file for each source code file. The header file defines data structures and declares functions. The source code file contains the source code for the functions you declared in the header file. In most cases you want to keep the checkbox selected. Because Xcode creates header files when it creates a source code file, you shouldn't need to explicitly create a header file often.

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 option to determine which project you want to add the file to. By default it's the current project, but you can choose another project you have open in Xcode or choose not to add it to any project.

Click the Finish 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.

Creating Your Own File Templates

Xcode has many file types to choose from, but Xcode's file types may not suit your needs. You may write code in a language that Xcode doesn't include in its new file list. You may want header files from additional frameworks to be included when you create a new file. Create a file template when Xcode's file templates aren't right for your programs. To create a file template:

1. Create a folder to hold the template. Give the folder the extension `.pbfiletemplate`.
2. Choose File > New File and add an empty file. The empty file is in the Other group. Click the Next button in the New File Assistant.
3. You'll be asked to name the file and for the location to save the file. Save the file in the folder you created in Step 1. The file should have the extension you want files of this type to automatically have. If you were creating an Objective-C++ file template, you would give the file the extension `.mm`.
4. If you have an Xcode project open, the file will be set to be added to the project. You don't want the file to be added to the project. Choose none from the Add to Project pop-up button. Click the Finish button.
5. If you want to include a header file with the implementation file, create a header file as well.
6. Put what you want in the file. You may want to look at Apple's file templates to give you an idea of what to put in the file. You can find the templates at `Developer/Library/Xcode/File Templates` and `Developer/Platforms/iPhoneOS.platform/Developer/Library/Xcode/File Templates`.

Before you can add your template to Xcode's file template list, you must add a property list file named `TemplateInfo.plist`. To create the property list file:

1. Launch the Property List Editor program.
2. Click the Add Item button twice.
3. Give one item the name Description and the other item the name `MainTemplateFile`.
4. The value of Description is the description you want to appear in the New File Assistant window.
5. The value of `MainTemplateFile` is the name of the file template you created.

If you included a header file, click the Add Item button a third time. The property name should be `CounterpartTemplateFile`, and the value should be the name of the header file you created. Save the property list file in the folder containing your file template and give the file the name `TemplateInfo.plist`. Move your file template folder to the following location:

```
/Library/Application Support/Developer/Shared/Xcode/File  
Templates
```

This folder contains all user-created Xcode file templates. If you haven't created any project or file templates, you will have to manually create some of the folders in the path to the file templates. If you want to create any grouping folders for your templates, create them in the File Templates folder and drag your file template folder into the grouping folder. After moving the file templates to the proper location, you will be able to create these files in Xcode by choosing File > New File. Your templates will appear in the User Templates section.

Fixing the Copyright Notice

If you look at the copyright notice of 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 your company name for the copyright notice. To change the company name, open System Preferences and click the Accounts 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, choose `Project > Edit Project Settings`. Click the General tab in the inspector. Use the Organization Name 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 `Project > Add to Project`. A dialog box opens that lets you find the files you want to add. After selecting files to add, a sheet like Figure 1.10 will open. Click the Add button to add the files.

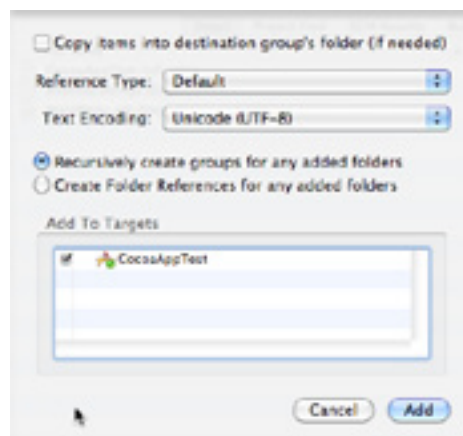


Figure 1.10

Adding files to project

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. The Reference Type determines how Xcode stores the file's location. There are six options.

- By group, which means Xcode uses the file's group in the Groups and Files list to store the file's location.
- By 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.
- By build product, which means Xcode uses the folder where it places build products, such as executable files and libraries, to store the file's location.
- By Xcode folder, which means Xcode uses the folder where you installed Xcode to store the file's location.
- By Current SDK , which means Xcode uses the folder where you installed the current SDK to store the file's location.

The default reference type for files is by group. The reference type options matter if you're going to share your projects with other people. If you referred to your files according to the absolute path on your computer, chances are high that other people using your project would have a different path to the files. In this case Xcode would be unable to find the files and the project would not compile.

If you're adding an entire 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 Groups and Files list for each folder you add. Recursively create groups when you need to access the individual files of the folder in Xcode. If you're adding a folder of source code files, you want to recursively create the group because you want to be able to examine and edit the source code files.

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

Source Trees

There can be another reference type for files you add to a project: by source tree. 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:

1. Open Xcode's preferences panel.
2. Click the Source Trees button in the preference panel's toolbar.
3. Click the + button to add a source tree.
4. Click the OK button.

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.

Adding Frameworks

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 application project. Many popular Apple technologies, such as QuickTime, OpenGL, and Core Audio, require you to add a framework to your project.

The easiest way to add a framework to your project is to select the Frameworks folder from the Groups and Files list (click the disclosure triangle next to the project at the top of the Groups and Files list if the Frameworks folder isn't visible) and choose Add > Existing Frameworks from the Action menu (right-click to display the Action menu). A sheet opens with a list of frameworks and libraries. Select the frameworks you want to add, and click the Add button.

Organizer

The Organizer lets you easily access folders you use often. How useful you find the Organizer is a matter of personal preference, but you can do the following things with the Organizer:

- Quickly open Xcode projects.
- Open and edit source code files without opening the project.
- Build, clean, and run Xcode projects without opening them.
- Write programs without creating an Xcode project.
- Create Java projects.

The ability to write programs without creating a project helps when you're not using Xcode's build system to compile your program. There are two common situations where you wouldn't use Xcode's build system. The first situation occurs when you're writing programs for multiple operating systems. You would want to write a build script that would work on each operating system. The Organizer lets you use your build script.

The second situation occurs when you're writing code in an unsupported language. Suppose you want to use Xcode to write Ruby on Rails applications. There's not much benefit in creating an Xcode project because Xcode doesn't support Ruby on Rails. By using the Organizer, you can use Xcode to write your code and supply your own scripts to build the application.

Opening the Organizer

Choose Window > Organizer to open the Organizer, which you can see in Figure 1.11.

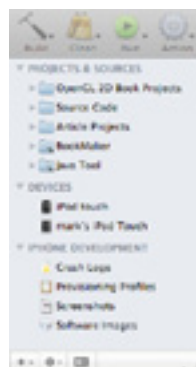


Figure 1.11

Organizer

At the top of the Organizer are four buttons: Build, Clean, Run, and Action. The Build button builds the final product (application as an example) from your code. The Clean button erases anything you previously built. The Run button runs your program. Read Chapter 4, “Building Projects” for more information on building, cleaning, and running. If you have an Xcode project, the Build, Clean, and Run buttons will build, clean, and run your project in either the Debug or Release build configuration.

The Action button is for general actions not related to building, cleaning, or running. A general action can be used for any task. You can add shell scripts and Automator actions to define your own Organizer actions. Refer to the section “Adding Your Own Actions” later in this chapter for more information.

At the bottom of the Organizer are three buttons. The first button lets you add files and folders to the Organizer. The second button is the Action menu, which lets you do a variety of things, including taking a snapshot, removing the folder from the Organizer, and opening the folder in the Finder. The last button toggles an editor in the Organizer.

Adding Folders to the Organizer

To make use of the Organizer, you must first add folders to it. The easiest way to add a folder to the Organizer is to drag the folder from the Finder to the Organizer. Alternatively, you can click the Add Item button at the bottom of the Organizer and choose Add Existing Folder.

When adding a folder to the Organizer, keep in mind that the Organizer treats that folder as a single unit. If you’re adding a folder that contains a single Xcode project, treating the folder as a single unit is no problem. But if the folder you added has lots of subdirectories, the Organizer’s behavior can cause problems.

Suppose you have a folder titled Source Code you use to store your Xcode projects. If you add the Source Code folder to the Organizer, all the projects in the folder are added to the Organizer. This simplifies adding a bunch of projects to the Organizer, but you lose some flexibility. You can’t remove the individual projects from the Organizer, only the Source Code folder. If you take a snapshot, Xcode takes snapshots of all the projects in the Source Code folder. If you want greater flexibility, add each project folder to the Organizer individually, which isn’t a problem if you drag them from the Finder. Double-click your Source Code folder, select the project folders, and drag them to the Organizer.

After adding folders to the Organizer, use the disclosure triangles to navigate the folders. Double-click a project file to open it. Double-click a source code file or text file to open it in a separate editor window.

Viewing Source Code Files in the Organizer

Opening a new window every time you open a file in the Organizer creates a lot of window clutter. To reduce the clutter, open the Organizer's editor by clicking the right button at the bottom of the organizer. The Organizer's editor lets you view and edit source code without having to create a new window. To view a file in the editor, select the file in the Organizer.

Building and Running Projects

If you create an Xcode project, building a project with the Organizer is simple. Select the folder containing the project. Click the Build button and hold the button down. A menu will open with items to build the project in Debug and Release configurations. Running a program is similar to building except you click the Run button instead of the Build button.

If there is no Xcode project in the folder you select, you'll notice no menu items to build or run your project. The only option will be to edit actions. If you choose not to use an Xcode project, you will have to choose Edit Actions to build, run, or clean your program.

Adding Your Own Actions

The first decision to make is the action you want to create. There are four types of actions: build, clean, run, and general. If you're not sure, go with a general action because a general action can do anything you need.

After deciding what type of action to create, click the appropriate button on the toolbar and hold the mouse button down. A list of options will open. Choose Edit Actions.

Click the + button to add an action. You can add a new shell script, an existing script file (shell script or AppleScript), an Automator action, or a separator. Separators are useful if you have lots of actions. You can use the separators to group actions.

Specifying a Directory

All Organizer actions have a Directory pop-up menu. It sets the working directory for the action. There are five directory choices.

- Selection, the directory selected in the Organizer.
- Top Level Organizer. At the top of the action window is a path control that represents the directory hierarchy. The Top Level Organizer is the leftmost folder in the hierarchy.
- Defining Organizer Item, the directory that created the action.
- Home Directory, the user's home directory: `/Users/username`.
- File System Root, the `/` directory.

If you're not sure what to choose, go with Selection.

Build Actions

There isn't much to specify for a build action. Supply the script or Automator workflow.

Clean Actions

A clean action takes the same arguments as a build action. Supply the script or Automator workflow.

Run Actions

There are three things to specify for a run action: command, arguments, and debugger. If you use an Automator workflow, the workflow is the command.

For an interpreted language like Ruby or Python, the command will be the language interpreter and the arguments will be the files to run in the interpreter. For a command-line C or C++ program, the command will be the name of the executable file and the arguments will be any command-line arguments you want to supply to the program.

There are three debugger choices: GDB, Java Debugger, and None. Choose GDB for C, C++, and Objective-C programs. Choose Java Debugger for Java programs. Choose None for all other languages.

General Actions

General actions are more flexible than the other actions. You specify the input, tell Xcode what to do with the output, and tell Xcode how to handle errors.

The input options are limited. The input can either be Selection or No Input.

There are five options to handle output: discard it, display it in an alert, place it on the clipboard to be pasted by another application, open it in a new document, and open it as HTML.

There are four options to handle errors: ignore them, display them in an alert, place them on the clipboard, and merge them with the output.

If you want to see some examples of general actions, choose Scripts > Edit User Scripts. The Scripts menu has a figure of a scroll instead of the word Scripts. Xcode comes with over 30 scripts for you to examine.

Removing Folders from the Organizer

To remove a folder from the Organizer, select it and choose Remove From Organizer from the Action menu at the bottom of the Organizer. If there is no Remove From Organizer item in the Action menu, you most likely have chosen a folder that is a subfolder of a folder you added to the Organizer. In this case, you won't be able to remove the folder. Only the folder you added can be removed.

Organizer for iPhone Apps

For those of you writing iPhone applications, the Organizer displays four additional items.

- Devices
- Archived Applications
- Developer Profile
- Device Logs
- Provisioning Profiles
- Screenshots
- Software Images

Devices

When you connect your iPhone to your Mac, the iPhone will appear in the Devices section of the Organizer. The first time you connect the iPhone, there will be a gray dot next to the device. This means the iPhone is connected, but isn't available for development. Select the device and an option will appear to use the device for development.

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

Archived Applications

The Archived Applications section lists the project builds you archived when choosing Build > Build and Archive. An archived application is a compressed archive of an iPhone application, similar to a zip archive. From this section you can validate your application, deliver copies of the application to beta-testers, and submit the application to iTunesConnect for placement in the App Store.

Developer Profile

The Developer Profile section has a list of your code signing certificates and provisioning profiles. The main use of the developer profile is to transfer your iPhone 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.

Device Logs

Clicking the Device Logs item shows a list of crash logs and other logs for the device. 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 the iPhone application you're developing crashes, drag the crash log to the Organizer. Xcode will add symbols (variable, class, and function names) to the crash log to make debugging easier.

Provisioning Profiles

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

Screenshots

The Screenshots section shows screenshots of your iPhone applications. There are two panes. The left pane has all the screenshots you've taken. Selecting a screenshot shows a full-size version of it on the right side. Clicking the Save as Default Image button makes that screenshot the default image for a project. When you click the Save as Default Image button, a window opens with a sheet of open projects. Select a project and click the OK button. The screenshot gets added to the project's Resources folder.

To take a screenshot of your iPhone, select the device from the devices list. Click the Screenshots tab. Click the Capture button. The screenshot appears in the right pane. If nothing appears, click the Capture button again.

Software Images

When you upgrade your iPhone to the latest version, 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.