Chapter 1

Xcode

Many computer books use Chapter 1 to cover introductory material. *Xcode Tools Sensei* is not one of those books. I want you to start learning immediately. After reading this chapter you'll know how to create a project, add files to your project, edit source code, model data, read developer documentation, configure the compiler, and build your project into a working Mac OS X program.

Creating a Project

Every program you write with Xcode requires a project, no matter how small the program is. An Xcode project contains your program's source code files and other files Xcode needs to build a working program, such as Interface Builder nib files. To create an Xcode project, choose File > New Project, and the Project Assistant window opens. Xcode has the following project categories:

- Action
- Application
- Audio Units
- Bundle
- Command-Line Utility
- Dynamic Library
- External Build System
- Framework
- J2EE
- Java
- Kernel Extension
- Standard Apple Plug-Ins
- Static Library

I will go into greater detail on the types of projects shortly, but most of you will be making application projects. 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:

- An Objective C source code file, `main.m`.
- The Cocoa, AppKit, and Foundation 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.
- A nib file, `main.nib`, 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 Carbon application is going to have the Carbon and Application Services frameworks instead of the Cocoa, AppKit, and Foundation frameworks. Java applications (the ones in the Java section of the Project Assistant Window) won't have any nib files and will have a Java source code file. Who would have thought that Xcode would do something as crazy as adding a Java source file to a Java project? 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 an 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 window on the screen.

Action Projects

Action projects create Automator actions. An *action* is a loadable bundle that performs one task. Create workflows by linking actions together. Use workflows to automate tasks in your application and other applications. You could use Automator with iTunes to transfer songs from a CD to an iPod. Apple introduced Automator in Mac OS X 10.4; older versions of Mac OS X cannot use Automator.

You can write Automator actions using AppleScript, Objective C, or shell scripting. Shell script action projects were introduced in Xcode 2.2. Xcode 2.2 also added the definition bundle project. Definition bundle projects allow you to define new data types for Automator actions.

Application Projects

The most common projects you will be creating will be application projects. Application projects create an executable file you can launch from the Finder. The type of application project you choose depends on the language you want to use. You can create Carbon applications in C or C++, or you can create Cocoa applications using AppleScript, Java, or Objective C. Apple stopped adding features to the Java version of Cocoa in Mac OS X 10.4 so I would advise against writing new Cocoa applications in Java. Improvements Apple makes to Cocoa will not make their way into the Java version.

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. A word processing program is an example of a document-based application; you can have five documents open at once. iTunes is an example of an application; you can't open five windows with each one playing a different song. If your application creates a new window when the user chooses File > New, you should create a document-based application.

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. Apple introduced Core Data in Mac OS X 10.4; older versions of Mac OS X can't use Core Data.

Audio Units Projects

Xcode 2.2 added audio units projects. The projects let you create Core Audio audio units. *Audio units* are software components that work with audio data. Audio units let you create plug-ins for audio applications and create your own sound effects.

Xcode provides an audio unit effect component project that creates an audio unit. There are two additional projects that include a view with the audio unit, one using Cocoa and the other using Carbon.

Bundle Projects

Bundles are directories of files that appear to the user as one file. They can contain executable files, images, sounds, strings, resource files, nib files, libraries, and frameworks. The beauty of bundles is that you don't have to worry about the user deleting or renaming files your program needs to run.

If you look at the Mac OS X documentation included with the Xcode Tools, you'll discover that applications and frameworks are also bundles. This information can cause confusion. When would you use a bundle project instead of an application or framework project? 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, including Web browsers (Safari), graphics programs (Photoshop and Maya), and development environments (REALbasic).

You can create bundles using the Carbon, Cocoa, and Core Foundation frameworks. Some versions of Xcode split the bundle projects into Bundle and Loadable Bundle sections.

Command-Line Utility Projects

Command-line utility projects create programs that run from the command line instead of the Finder. If you're learning C or C++, command-line utility 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 Aqua. By using a command-line utility project, you can learn the language without having to deal with the complexity of writing Mac programs. The Standard Tool project creates a C language tool while the C++ Tool project type creates a C++ tool.

You can also create command-line utility projects that use the Core Services, Core Foundation, and Foundation frameworks. The lowest level of Mac OS X is Darwin, which is a flavor of Unix. The Core Services framework sits on top of Darwin, and higher-level frameworks like Carbon and Cocoa sit on top of Core Services. Figure 1.1 shows the relationship between the 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.

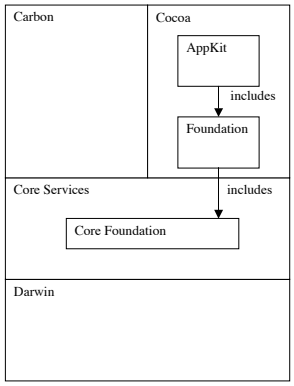


Figure 1.1

The relationship between the basic Mac OS X frameworks. The AppKit framework's header file includes the Foundation framework's header file. The Foundation framework's header file includes the Core Foundation framework's header file.

One part of Core Services is the Core Foundation framework, which defines basic data types, such as numbers, strings, arrays, and dates. Core Foundation provides support for plug-ins, bundles, user preferences, property lists, and XML. Because Core Services includes Core Foundation, you can use a Core Services tool project to write Core Foundation code. Only use a Core Foundation tool project if you're sure you don't need to use other parts of the Core Services framework.

Foundation tool projects use the Objective C Foundation framework that is part of the Cocoa framework. Use a Foundation tool project to write an Objective C program without a graphical user interface.

Dynamic Library Projects

Dynamic libraries are pieces of code an application loads when it's running. You can create dynamic libraries for Mac OS X using the Carbon and Cocoa frameworks. The BSD dynamic library project lets you write a dynamic library using the BSD Unix APIs. If you're writing a dynamic library you want to use on Linux or Unix operating systems, create a BSD dynamic library project.

External Build System Projects

External build system projects (Older versions of Xcode call them GNU make projects) use a program other than Xcode to build the projects. Use an external build system project when writing a program in a language other than the ones Xcode natively supports: 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.

Suppose you're writing a Python program. You can write all your Python code in Xcode and build the program from Xcode by creating an external build system project. All you have to do is tell Xcode to build your program with `pythonw`, the Python interpreter. When you build your project in Xcode, it uses `pythonw` to do the building, letting you avoid the command line.

Framework Projects

Frameworks are a special kind of bundle. They include header files, resource files, dynamic shared libraries, and reference documentation. Unlike other types of bundles, frameworks can include multiple versions in a single bundle. By storing multiple versions in a bundle, a developer can make improvements to a framework without breaking applications that use an older version of the framework.

Use framework projects if you want other programmers to use your code when creating their applications. Suppose you wrote a class library to handle networking between different operating systems. By writing your class library as a framework project, other programmers could use your code to handle networking in their programs. All they would have to do is add your framework to their projects and include the framework's header files in their source code files.

J2EE Projects

J2EE projects use the Java 2 Platform, Enterprise Edition (J2EE). Use J2EE to develop component-based large-scale enterprise applications in Java, including applications that run on the Internet. There are three types of J2EE projects.

- Enterprise Java Beans (EJB) module projects create J2EE business components.
- Enterprise application projects create J2EE business applications, which can consist of multiple EJB and Web modules.
- Web module projects create Web applications.

All three types of J2EE projects use Ant to build the project and use the XDoclet code generation engine.

Java Projects

The main advantage of using Java to write programs is the same code runs on multiple operating systems. Xcode lets you write Java programs that will run on Mac OS X as well as other operating systems like Windows and Linux. Xcode provides many options for writing Java programs.

- Ant projects use Ant to build the program instead of the Java compiler. Ant is a cross-platform build tool that lets programmers build Java programs on any operating system and IDE that supports Ant, such as Xcode. Ant makes open source projects easier to build because it works on multiple operating systems and development environments.
- Abstract Window Toolkit (AWT) projects use the AWT framework. AWT is a class library for writing Java programs with graphical user interfaces.
- Swing projects use the Swing framework to write Java programs with a graphical user interface. Swing inherits from the AWT library, which means the Swing library includes the AWT library as well.
- Java Native Interface (JNI) 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.
- Java tool projects produce programs without a graphical user interface. A Java tool project is the right choice if you're learning Java.

If you use AWT or Swing, you can create applets or applications. Applets are programs that will run in another application. Many websites use Java applets that run in your web browser. Java applications are no different than other Mac OS X applications. You can place them in the Dock and launch them from the Finder.

Assuming you want to write a program with a graphical user interface that runs on multiple operating systems, you must decide whether to create a Swing project or an AWT project. In most cases you'll want to create a Swing project. Because Swing is built on top of AWT, you can create a Swing project and use AWT code in it. Only create an AWT project if you're absolutely sure you won't need to use any Swing code in your program.

What project type do you choose if you want to use Ant to build your project, but also want to use the AWT or Swing frameworks? Use an Ant project, and add the AWT and Swing frameworks in your source code.

```
import java.awt.*;
import javax.swing.*;
```

Xcode 2.2 added two Java project types: signed applets and Web Start applications. Signed applets are applets that contain a digital signature. Web Start is a technology that lets users easily download and launch Java applications from the Internet.

Kernel Extension Projects

Apple's documentation discourages you from writing kernel extensions so you might be surprised that Xcode includes a project template for kernel extensions. 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. You're more likely to create an IO Kit Driver project, which you use to write device drivers for computer peripherals like printers, joysticks, and disk drives.

Standard Apple Plug-In Projects

Use the standard Apple plug-in projects to write plug-ins for Apple programs. You can write plug-ins for Address Book, Core Image, Installer, Interface Builder, Sherlock, System Preferences, and Xcode.

You can also write screen savers, metadata importers, and sync schemas. Metadata importers extract metadata from files. Sync schemas synchronize data between your program and other applications. Core Image and Installer plug-ins, metadata importers, and sync schemas are not available in versions of Xcode prior to Xcode 2.0.

Static Library Projects

Static libraries are pieces of code your application links against when compiling your code. You can create static libraries using the Carbon and Cocoa frameworks. The BSD static library project lets you write a static library using the BSD Unix APIs. If you're writing a static library you want to use on Linux or Unix operating systems, create a BSD static library project.

Creating Your Own 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. To create a project template:

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

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

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

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

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

I'm going to cover the toolbar and the Groups and Files list shortly. The favorites bar is initially invisible. Choose View > 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 and documentation pages. To add an item to the favorites bar, select the item and drag it to the favorites bar.

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. Clicking the Editor button in the toolbar 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.

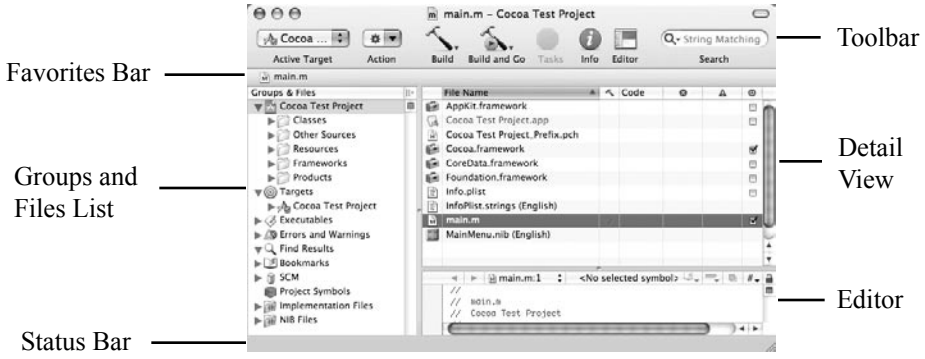


Figure 1.2

Project window.

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:

- The Active Target menu lets you select which of your project's targets is the current one.
- The Action menu contains frequently used commands. What appears in the menu depends on what you select in the Groups and Files list.
- The Build Active Target button builds your project if you click it. If you hold the button down, a menu opens, giving you the option of cleaning your target. Cleaning the target removes all the object code, forcing you to recompile all your source code files.
- The Build and Go button builds your project and runs the program if you click it. If you hold the button down, a menu opens. From it you can choose to build and debug your program, run your program without building it, and debug your program without building it.
- The Tasks button will stop any programs you have executing in Xcode.
- The Info button opens the selected item's information panel. You will use this button to tweak compiler settings, which is a topic I will cover later in the chapter.
- The Editor button replaces the project window's detail view with an editor so you can edit your source code files. If you want to have only one window open, clicking the editor button gives you more room for editing. Click the Editor button a second time to restore the detail view.
- The Search field lets you filter listings in the project. If you wanted to see only header files in the project window, you would type `.h` in the search field.

If those items aren't what you use most, you can customize the toolbar to suit your needs. To customize the toolbar, choose View > Customize Toolbar. A window like Figure 1.3 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.

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. Control-clicking 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.

Figure 1.3

Customize toolbar window.

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 1.4 shows what the detail view looks like when you select the project name from the Groups and Files list.

The detail view displays seven columns of 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.
- 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.
- 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.

Double-clicking a source code file or a property list file opens the file in a separate window. Double-clicking a nib 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, click the Editor button in the project window toolbar. It will add an editor pane to the project window so you can edit source code. 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.

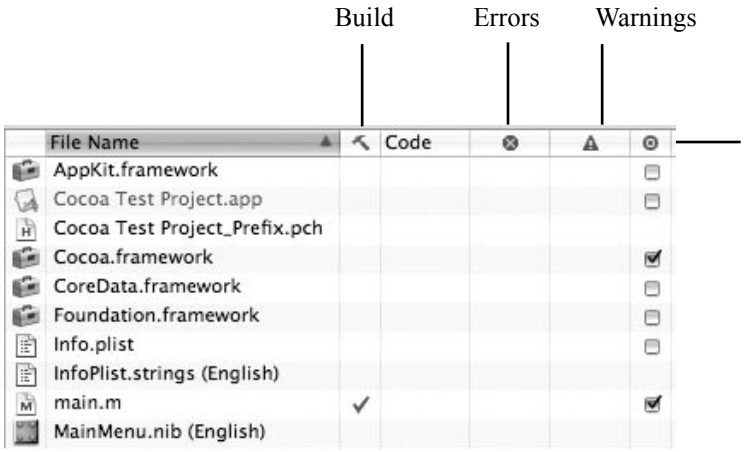


Figure 1.4

Project window 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" explains how to add targets to a project. If you have multiple targets in your project, use the Active Target pop-up menu to choose the active target. The section "Working with Targets" 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. 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 information panel. 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 1.5, 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.

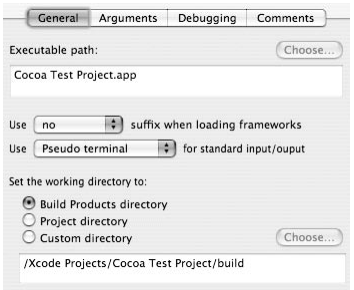


Figure 1.5

Executable general panel.

For bundled applications the path is `ExecuableName.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. Use the profile variant if you're using `gprof` to profile your program.

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 run log 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 run log. 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 programs do not deal with working directories. By default the working directory 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. Tool 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 debugging next chapter, in which I cover the debugging panel in more detail.

Errors and Warnings

The Errors and Warnings group lists any compiler errors or warnings in your project. This group will be empty until you compile the files in your project. Hopefully the group will remain empty after compiling. For each error and warning in your program, Xcode reports the following information:

- Was it an error or a warning? Errors prevent the program from being built. Warnings alert you to possible problems in your program.
- The error or warning message.
- The file where the error or warning occurred and the line number.

Double-clicking a listing in the Errors and Warnings group takes you to the line of code where the error or warning occurred. From there you can fix the problem.

Find Results

The Find Results group lists the results of searches you made for this project. It's helpful if you search for a particular term in your project, then 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, then choose File > 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 11 for more information on using version control with Xcode.

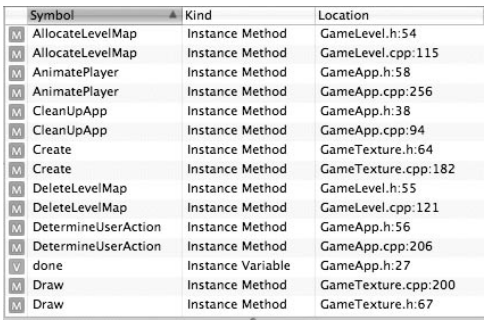
Project Symbols

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.

Figure 1.6 shows a sample listing of project symbols. The 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 `#define` statement. C programs use macros to define constants.
Type	If you use the `typedef` 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.



Symbol	Kind	Location
AllocateLevelMap	Instance Method	GameLevel.h:54
AllocateLevelMap	Instance Method	GameLevel.cpp:115
AnimatePlayer	Instance Method	GameApp.h:58
AnimatePlayer	Instance Method	GameApp.cpp:256
CleanUpApp	Instance Method	GameApp.h:38
CleanUpApp	Instance Method	GameApp.cpp:94
Create	Instance Method	GameTexture.h:64
Create	Instance Method	GameTexture.cpp:182
DeleteLevelMap	Instance Method	GameLevel.h:55
DeleteLevelMap	Instance Method	GameLevel.cpp:121
DetermineUserAction	Instance Method	GameApp.h:56
DetermineUserAction	Instance Method	GameApp.cpp:206
done	Instance Variable	GameApp.h:27
Draw	Instance Method	GameTexture.cpp:200
Draw	Instance Method	GameTexture.h:67

Figure 1.6

Project symbols listing.

Smart Groups

Smart groups do not have the name Smart Groups. They're folders that group files automatically 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 `.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 File > 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 information panel opens, as you can see in Figure 1.7.

From the information panel 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.

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|scpt)$
```

This expression says that any file that contains the characters `.c`, `.cpp`, `.C`, `.CPP`, `.m`, `.mm`, `.java`, `.sh`, and `.scpt` appears in the group. These characters just 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.

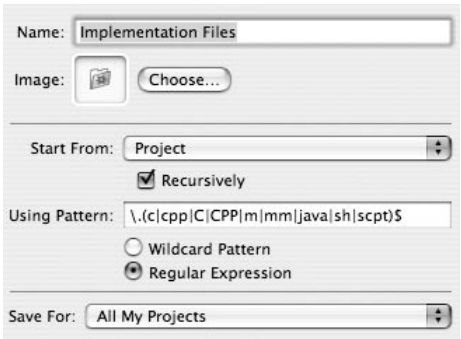


Figure 1.7

Smart group information panel.

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. You can create new files or add existing files to your project.

Adding New Source Code 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. Creating a new file consists of two steps.

- Choosing a file type.
- Naming the file.

Enterprise Java Beans and servlet files have a third step. The third step involves setting properties for the file you're creating.

Choosing a File Type

When you create a new file by choosing File > New File, the New File Assistant dialog box opens. The dialog box shows a list of files you can create. 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. If you choose to create a J2EE Enterprise Java Beans file, you must choose the type of bean you want to create before you can name the file.

The file list contains nine types of Cocoa files: four Java files and five Objective C files. You can create Objective C test case files, plain class files and subclasses of `NSDocument`, `NSView`, and `NSWindow`. If you're not sure what type of Cocoa class you should create, choose a Cocoa class file instead of one of the subclass files.

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. Figure 1.8 shows what the dialog box looks like. 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`.

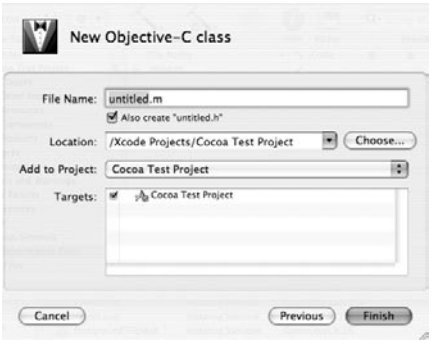


Figure 1.8

Xcode's New File Assistant.

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. Enterprise Java Beans and servlet files have another dialog box of properties to set before you can click the Finish button. Figure 1.9 shows an example of the file Xcode creates. It includes 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) Choose File > New Empty File to create an empty file.
2) If you want to include a header file with the implementation file, create a header file as well.
3) Put what you want in the file.
4) Save the file. It 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`.
5) Move the file and any header file into its own folder. The folder should have the extension `.pbfiletemplate`.

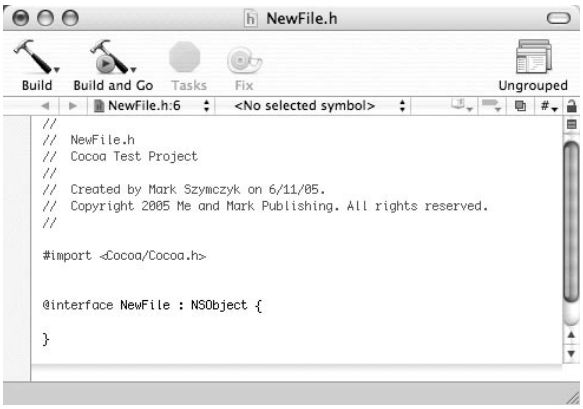


Figure 1.9

A new file that Xcode creates.

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 New Root button.
3) Click the disclosure triangle next to Root.
4) Click the New Child button twice.
5) Give one child the name `Description` and the other child the name `MainTemplateFile`.
6) The value of `Description` is the description you want to appear in the New File Assistant window.
7) The value of `MainTemplateFile` is the name of the file template you created.

If you included a header file, click the New Child button a third time. The property name should be `CounterpartTemplateFile`, and the value should be the name of the header file you created.

After creating the `TemplateInfo.plist` file, move it to the folder containing your file template. Move the folder to `/Library/Application Support/Apple/Developer Tools/File Templates`. This folder contains all the Xcode file templates. Now you can create these files in Xcode.

Fixing the Copyright Notice

If you look at the copyright notice of the files you create, it will probably 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 you must edit Xcode's preferences file. The preferences file resides in the folder `/Username/Library/Preferences` and has the name `com.apple.Xcode.plist`. Double-click the preferences file to open it in the Property List Editor program. Select Root and click the disclosure triangle next to Root to see all the preferences.

The text for the button in the upper left corner should change to New Child. Click the New Child button to create a new preference. Type the name `PBXCustomTemplateMacroDefinitions`. Select this preference. Its class should be String. Click on the word String to open a menu; change the class to Dictionary. Changing the class adds a disclosure triangle to the left of the `PBXCustomTemplateMacroDefinitions` property.

Click the disclosure triangle next to `PBXCustomTemplateMacroDefinitions`. Click the New Child button to add an item to the dictionary. The item's initial name is New Item. Change it to `ORGANIZATIONNAME`; make it all uppercase letters. The item has class String, which is what you want. The item's value is currently empty. Double-click the value to type in your company name. If you don't work for a company, type your name. Figure 1.10 shows you what you need to type. Save the file. Quit Xcode and relaunch it to have the changes you made take effect. If you don't quit Xcode, it uses the old settings, which means the files you create will say MyCompanyName instead of your company's name.

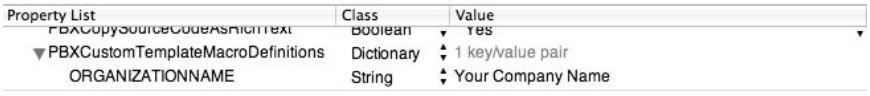


Figure 1.10

Adding your company name to Xcode's property list so your company name appears in source code files that Xcode creates.

Adding Files You've Already Created

Having to write every line of code from scratch for every project would get tiresome quickly. Adding files you created for previous projects makes developing new programs easier. 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 1.11 will open. Click the Add button to add the files.

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 four 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.

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.

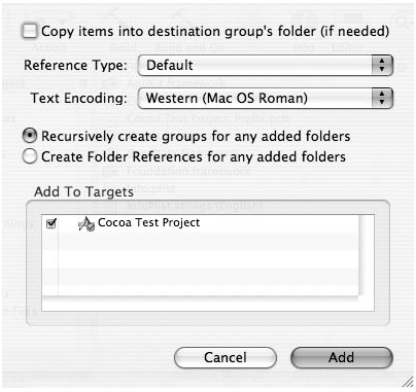


Figure 1.11

Add File dialog.

Source Trees

There can be a fifth 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) Choose Xcode > Preferences to open Xcode's preferences panel.
2) Select Source Trees from the preferences panel.
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 Carbon or Cocoa frameworks. Many popular Apple technologies, such as QuickTime, OpenGL, and Core Audio, require you to add a framework to your project. Adding a framework to your project is identical to adding previously created files. Choose Project > Add to Project. You can find most of Apple's frameworks in `/System/Library/Frameworks`. After selecting the frameworks you want to add, the sheet shown in Figure 1.11 opens. Click the Add button to add the framework to your project.

Editing Source Code

When you develop software you spend a lot of time writing and editing source code. This section covers Xcode's features for editing source code.

The Editor Window

Double-clicking a source code file in the project window opens an editor window. Figure 1.12 shows a typical editor window, which contains five parts.

- Toolbar
- Status bar
- Navigation bar
- Editor
- Gutter

Toolbar

The toolbar, which runs along the top of the editor window, contains commonly used commands. The toolbar in Figure 1.12 contains the default toolbar items. The buttons on the left side of the toolbar have nothing to do with editing source code. The first two buttons deal with compiling and running your program. The Fix button works with debugging, and the Tasks button stops tasks like compiling, running, and debugging your program.

The Grouped button (Editing Mode button in some versions of Xcode) is the most interesting toolbar button for source code editing. The initial editing mode is single window. Single window editing mode means there's only one editor window open at a time. When you open another file Xcode places the new file's contents in the window. Clicking the Grouped button changes the editing mode to multiple window, and the button name changes to Ungrouped. When you use multiple window editing mode, each source code file opens in its own window.

Customizing the Toolbar

Choose View > Customize Toolbar to customize the editor window's toolbar. Make sure an editor window is the current window before choosing View > Customize Toolbar. A sheet will open in front of the window. Customizing the editor window toolbar is similar to customizing the project window toolbar; drag the icons you want to add from the customization sheet to the toolbar. Click the Done button to finish customizing.

Status Bar

The status bar doesn't do much when you're editing source code. It shows you the status of Xcode activities like building and debugging programs. The status bar can be in one of two places, depending on your version of Xcode. It can be below the toolbar or at the bottom of the window.

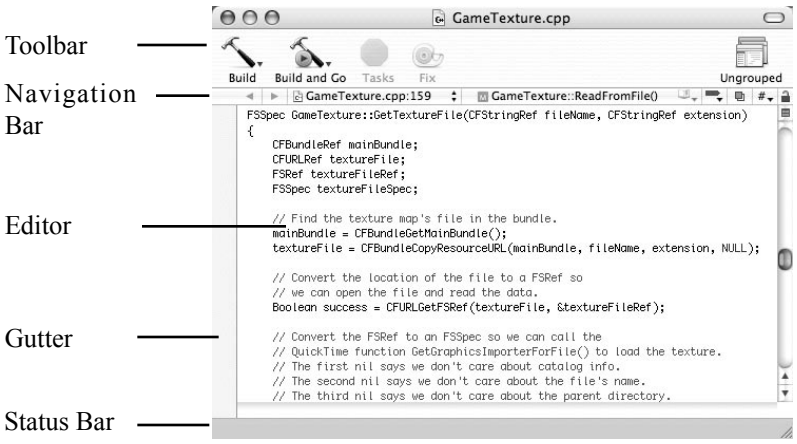


Figure 1.12

Xcode's editor window.

Navigation Bar

The navigation bar, shown in Figure 1.13, contains controls to quickly reach areas of the current file and quickly change files. It has four different areas. On the left edge of the bar are back and forward buttons, which work similarly to an Internet browser's back and forward buttons.

To the right of the back and forward buttons is the recently viewed files list. The list displays the file you're viewing and the line number. The pop-up menu next to the file name and line number shows the list of recently viewed files. The recently viewed files list helps when you edit a file, edit a few more files, and want to go back to the first file you were editing.

Next to the recently viewed files list is the function list. It shows the function you're at in the file. There's a pop-up menu containing all the functions in the file. Use the function list to reach a function in the file quickly.

At the right edge of the navigation bar are four buttons. Clicking the leftmost button lists the bookmarks you've set in the file. Clicking the second button lists the breakpoints you've set in the file. The third button is the Go To Counterpart button, which works with C-based languages. If the current file is an implementation file, clicking the Go To Counterpart button opens the header file for the current file. If the current file is a header file, clicking the Go To Counterpart button opens the implementation file. Clicking the fourth button lists all the files the current file includes. Selecting a file from the list opens that file.

Some versions of Xcode have a fifth button. Use this button to lock and unlock files. Locking a file prohibits anyone from editing the file.

Editor and Gutter

The editor is where you edit source code. You've used word processors and text editors before. Xcode's editor isn't radically different from editors you've previously used. The gutter runs along the left edge of the window. Its main use is for setting breakpoints, which I will discuss next chapter.

Code Completion

Code completion tells Xcode to finish the names of your program's functions and variable names, saving you from having to type the whole name.

Using the Completion List

To perform code completion, start typing the function or variable name and press the Esc key. A pop-up menu known as a *completion list* will open. The completion list contains all the possible matches. Select an entry from the completion list and press the Return key to complete the code. If you don't want to complete the code, press the Esc key to close the completion list.

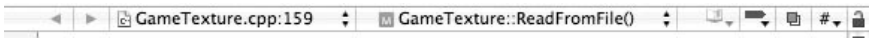


Figure 1.13

The editor window's navigation bar.

Cycling Through Completion Matches

An alternative method of code completion is to cycle through the possible matches instead of opening a completion list. Pressing Control-period displays the next match in the editor. Keep pressing Control-period until your desired match appears.

Customizing Code Completion

For code completion to work, you must have the Enable Indexing for all projects checkbox selected in Xcode's preferences panel, shown in Figure 1.14. Xcode creates a project index that keeps track of all the project symbols, such as variable names, function names, and constants. Xcode uses this index for many things, one of which is code completion. If you turn off indexing, you disable code completion as well.

Xcode has four checkboxes to customize how code completion works. When you select the Indicate when completions are available checkbox, Xcode underlines the text you're typing if the text has completions available.

Selecting the Show arguments in pop-up list checkbox tells Xcode to add arguments for each function that appears in the completion list. If you don't select the checkbox, the completion list shows the function name only. Adding arguments helps when you have functions with the same name but different arguments. Without the arguments you would have no way of knowing the function you were completing.

When you select the Insert argument placeholders for completions checkbox, Xcode inserts placeholders for the function's arguments when it completes a function. If you don't select the checkbox, Xcode completes the function name only. Here's how Xcode would complete the OpenGL function `glVertex3f()` with the function name only:

```
glVertex3f
```

And here's how Xcode would complete the `glVertex3f()` function with argument placeholders:

```
glVertex3f(<#GLfloat x#>,<#GLfloat y#>,<#GLfloat z#>)
```

If you tell Xcode to insert argument placeholders, press Control-slash to move to the next argument in the function. The Control-slash combination makes it easy to replace the argument placeholders with your own arguments.

Use the Automatically suggest pop-up menu to open the completion list automatically. Xcode can automatically open the completion list in two ways: open the list all the time or open the list when accessing your classes' members. You specify the amount of time that must pass before Xcode opens the completion list.

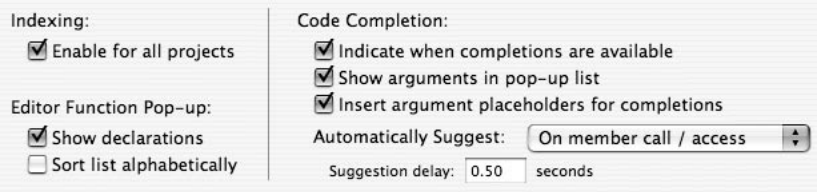


Figure 1.14

Code Sense preferences panel.

Getting Information About Functions and Variables

Command-double-clicking a symbol in your source takes you to that symbol's declaration point. For class variables Xcode opens the header file where you declared the variable. For functions you wrote, Xcode takes you to the function. For functions in external frameworks, Xcode opens the header file and shows you the declaration. The header files in Apple's frameworks contain documentation about the framework's functions. Command-double-clicking doesn't work for local variables.

Customizing Source Code Editing

To customize your experience editing source code, open the Xcode preferences panel by choosing Xcode > Preferences. The preferences areas of interest are the following:

- Text Editing
- Fonts and Colors
- Indentation
- Key Bindings

Text Editing Preferences

The text editing preferences, shown in Figure 1.15, is where you set miscellaneous text editing options. Some options you can set include the following:

- What pressing the Return key to end a line of code generates: a carriage return or a line feed. Mac OS X, Unix, and Windows end lines differently. Mac OS X generates a carriage return. Unix generates a line feed. Windows generates a carriage return and a line feed.
- Showing the gutter on the left side of the editor window.
- Showing line numbers in the editor.

Fonts and Colors Preferences

Xcode gives different areas of your code their own text color to make reading the code easier. Comments have green text. Numbers have blue text. Strings and language keywords (words used by the language such as `if`, `while`, `for`, and `int` in C) have red text. The syntax coloring preferences is where you set the text color, font, and point size for your code.

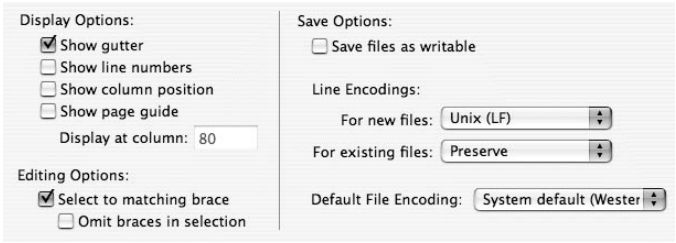


Figure 1.15

Text editing preferences.

Indentation Preferences

Programmers use indentation to make the program's logic clearer. Compare the following code snippets:

```
for (short row = 0; row < rowCount; row++) {
for (short column = 0; column < columnCount; column++) {
if (IsRectangleDirty(row, column)) {
DrawTile(row, column);
}
}
}
```

```
for (short row = 0; row < rowCount; row++) {
    for (short column = 0; column < columnCount; column++) {
        if (IsRectangleDirty(row, column)) {
            DrawTile(row, column);
        }
    }
}
```

The second snippet is easier to read because of the code indentation. The indentation preferences let you turn on automatic code indenting, which Xcode calls syntax aware indenting. When you type a statement that calls for indentation, such as `if`, `for`, and `while` statements, and press the Return key, Xcode indents the new line for you. If you enter the statement without a leading brace, such as the following:

```
if (x == 0)
    y = 0;
else
    y = 1;
```

Xcode's indenting is smart enough to know an `if` statement without a brace has just one statement following it. When you press Return after typing the line `y = 0`, Xcode will line up the cursor with the `if` statement so the `if` and `else` lines start at the same column.

Key Bindings Preferences

Use the key bindings preferences to set keyboard shortcuts for practically anything. You can set a keyboard equivalent for any menu item in the Xcode menu bar using the Menu Key Bindings tab. The Text Key Bindings tab lets you set keyboard equivalents for things like text editing, moving the cursor, and text formatting.

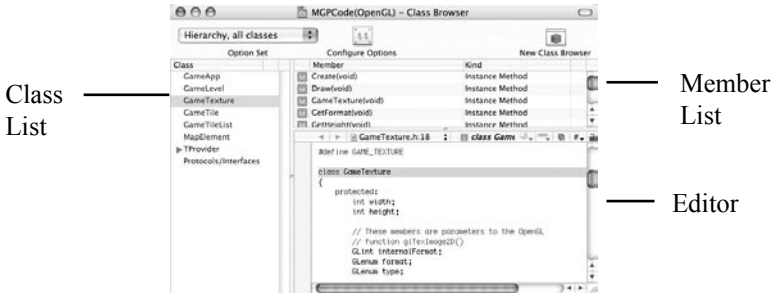


Figure 1.16

Class browser window.

Xcode comes with default key binding sets for Xcode, BBEdit, CodeWarrior, and MPW. If you're moving to Xcode from BBEdit, CodeWarrior, or MPW, the key binding sets will smooth the transition to Xcode. To create a custom set of key bindings, click the Duplicate button to create a copy of one of the default sets. Use the copy to bind keys.

Using Xcode's Class Browser

For those of you using object-oriented languages like C++, Java, and Objective C, Xcode has a class browser to examine your classes' members. Choose Project > Show Class Browser to open the class browser, which you can see in Figure 1.16. The class browser has three areas.

- Class list
- Member list
- Editor

If no classes appear in the browser, you may need to index your project. To index your project, select the project name from the Groups and Files list and click the Info button. The project's information panel will open. Click the General tab. Click the Rebuild Code Sense Index button to create the index.

Browsing Classes

The class list runs down the left side of the window. Classes you wrote appear in blue text. Classes defined in another framework, such as the built-in Cocoa classes, appear in black text. If a class has subclasses, that class will have a disclosure triangle next to it. Click the disclosure triangle to see the subclasses.

Selecting a class from the class list will do two things. First, Xcode fills the member list with the class's members. Second, Xcode opens the class's header file in the editor. Selecting a method from the member list takes you to the method's declaration in the header file.

Customizing What the Class Browser Shows

You can customize what appears in the class browser by clicking the Configure Options button in the class browser toolbar. Doing so will open the class browser configuration window, shown in Figure 1.17. Looking at the figure, you can see two columns of controls. The left column deals with what appears in the class list, and the right column deals with what appears in the member list. Read the next two sections to learn what you can customize for the class and member lists.

At the top of the configuration window is a pop-up menu containing saved configurations. A saved configuration saves your settings so you don't have to configure the class browser every time you use it. The class browser comes with four saved configurations, and you can add configurations to the list. Click the Add button to create a saved configuration. Name the configuration, use the configuration window to modify the class browser settings, and click the OK button. The configuration you created now appears in the Option Set pop-up menu in the class browser window.

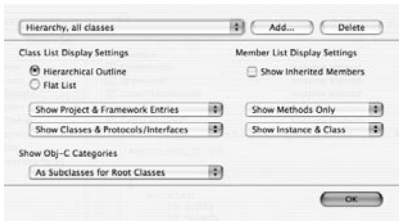


Figure 1.17

Class browser configuration window.

Customizing What Appears in the Class List

In the Class List Display Settings section there's a radio button group asking you whether you want the classes in your project to appear in a hierarchical or a flat profile. In a hierarchical profile, classes with subclasses have a disclosure triangle, which you click to see the subclasses. In a flat profile the classes appear in alphabetical order.

Below the radio button group are three pop-up menus. The first pop-up menu lets you decide whether you want only classes from your project, only classes from external frameworks (Cocoa or Carbon for example), or both types of classes to appear in the class browser. The second pop-up menu lets you decide whether you want only classes, only protocols and interfaces, or both to appear in the browser. Protocols and interfaces are different names for the same thing. Objective C uses the term protocol, and Java uses the term interface. A *protocol* is a collection of method declarations. These declarations are not part of a class. Any class can implement the methods in the protocol.

The third pop-up menu applies only to Objective C programs. It lets you determine how to show Objective C categories. *Categories* let you extend existing classes by adding methods to them, which is something you can't do in C++ and Java. By using categories you can add methods to the built-in Cocoa classes. You can show categories as subclasses of the class you're extending, subclasses for root classes, or show them merged into the class itself.

Customizing What Appears in the Member List

In the Member List Display Settings section there's a checkbox asking you if you want to show inherited members of a class. If you select this checkbox, the inherited members of the class appear in gray text in the member list. The non-inherited members appear in black text to distinguish them.

Below the checkbox are two pop-up menus. The first pop-up menu asks whether you want to show methods, data, or both in the member list. By default the class browser shows methods only.

The second pop-up menu asks whether you want to show the instances, show the class, or both. *Instances* are the class's data members and non-static member functions. If you choose class only, only static functions will appear in the member list. In Objective C code, only class methods will appear if you choose class only; no instance methods will appear in the member list. Because most functions in classes are non-static, you'll want to see instances in the class browser.

Using Xcode's Modeling Tools

Apple added visual modeling tools in Xcode 2.0, the version that ships with Mac OS X 10.4. There are two modeling tools: the class modeling tool and the data modeling tool. The class modeling tool works with C++, Java, and Objective C programs. The data modeling tool works with Cocoa applications that use the Core Data framework.

Modeling Classes

Xcode's class modeling tool shows the same kinds of information the class browser shows. You can examine your project's classes, view the members of each class, and see the inheritance relationships each class has. Why would you want to use the class modeling tool instead of the class browser? The class modeling tool has the following advantages: