Xcode Workspace Guide

Tools > Xcode

2009-01-06

Contents

Chapter 4   **The Text Editor   51**

4

Figures and Tables

Chapter 5   **Documentation Access   87**

Chapter 6   **Keyboard Shortcuts   99**

Chapter 7   **User Scripts   103**

8

Introduction

The Xcode workspace is made up of the windows you use regularly to develop products using the Xcode application. Such windows include the project window, text editor windows, the Documentation window, and others. Xcode lets you arrange the components of the project window and specify what documentation the Documentation window shows. When it comes to editing text files, especially source-code files, the Xcode text editor provides many features that facilitate editing code and access API reference directly from the editor quickly and accurately.

This document presents all the components that make up the Xcode workspace. If you're new to Xcode, you should read this document to familiarize yourself with these components and to learn how to arrange them to your liking. *What's New in Xcode* describes features introduced in the latest release of Xcode.

Organization of This Document

This document contains the following chapters:

- "The Project Window" (page 11) introduces project window components and describes the available project window layouts.

- "Project Organization" (page 29) provides tips for partitioning and arranging the code and resources for a product as you develop with Xcode.

- "File Management" (page 39) describes how to edit file, folder, and framework references in your project. It also describes how to change the way in which Xcode handles a file by changing its type, and how to control the way a file is displayed and saved, by changing the file encoding and line ending.

- "The Text Editor" (page 51) describes the Xcode text editor, shows how to open files in a standalone window or in an editor pane, and how to control the appearance of the editor.

- "Documentation Access" (page 87) discusses the documentation-viewing experience Xcode provides through the Documentation window and the Research Assistant.

- "Keyboard Shortcuts" (page 99) shows how to view and change the keyboard shortcuts for menu items and key-based actions.

- "User Scripts" (page 103) describes how to use predefined user scripts and how to create custom user scripts.

- "Resetting Xcode" (page 111) explains how to reset Xcode to its factory settings.

See Also

- *Xcode Overview* describes the Xcode IDE and its major components, and provides and overview of the development process.

- *Xcode Quick Tour for iPhone OS* showcases the tools used to develop iPhone applications.

- *Xcode Quick Tour for Mac OS X* provides a hands-on introduction to Xcode, Apple's comprehensive suite of software development tools for Mac OS X.

- *Xcode Project Management Guide* provides practical descriptions of the major development tasks developers perform with Xcode.

- *Xcode Source Management Guide* describes how to manage source changes using source control and snapshots.

The Project Window

The project window is where you do most of your work in Xcode. The **project window** displays and organizes your source files, targets, and executables. It allows you to access and edit all the pieces of your project. To work effectively in Xcode, you need to recognize the parts of the project window and understand how to use them to navigate your project's contents.

Of course, everyone has their own way of organizing their workspace. To help you be as efficient and productive as possible, Xcode provides several project window layouts. A project window layout specifies a particular arrangement for a project window, as well as ancillary task-specific windows.

This chapter introduces project window components and describes the available project window layouts. It also introduces other important Xcode windows and provides tips on using the Xcode interface to locate information on project items.

Project Window Components

Figure 1-1 identifies some of the components of the project window using the default layout. You can add, remove, and configure these components to your liking.

Figure 1-1   The default project window layout
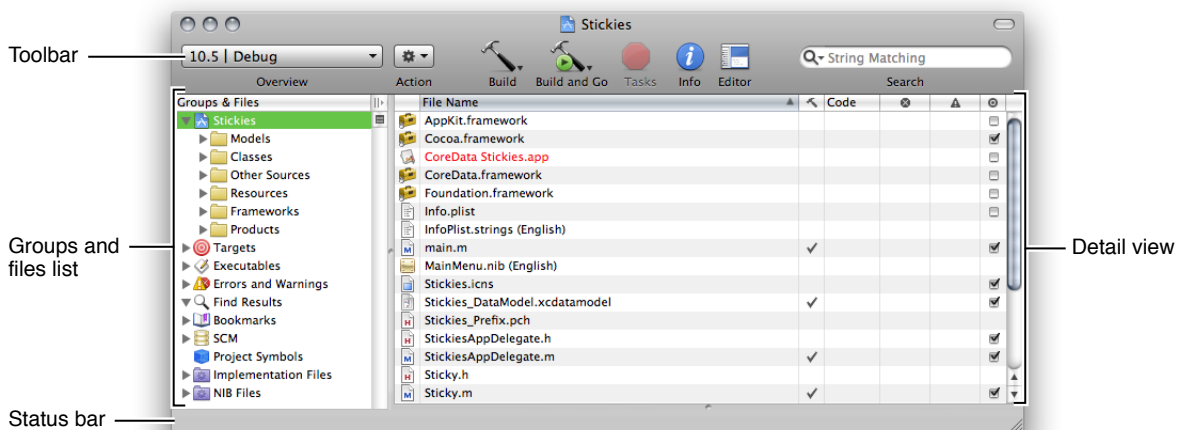
The project window contains the following key areas for navigating your project:

■   **Groups & Files list.** Provides an outline view of your project contents. You can move files and folders around and organize your project contents in this list. The current selection in the Groups & Files list controls the contents displayed in the detail view.

- **Detail view.** Shows the item or items selected in the Groups & Files list. You can browse your project's contents in the detail view, search them using the search field, or sort them according to column. The detail view helps you rapidly find and access your project's contents.

- **Toolbar.** Provides quick access to the most common Xcode commands.

- **Status bar.** Displays status messages for the project. During an operation—such as building or indexing—Xcode displays a progress indicator in the status bar to show the progress of the current task.

- **Favorites bar.** Lets you store and quickly return to commonly accessed locations in your project. To display the favorites bar, choose View > Layout menu > Show Favorites Bar. For details about using the favorites bar, see "Adding Items to the Favorites Bar" (page 35).

The project window also contains a text editor pane that lets you edit files directly in the project window. You can navigate through the views in a window, except the text editor pane, by pressing Tab.

For a description of each of the available project window layouts, see "Project Window Layouts" (page 20).

The Groups & Files List

The Groups & Files list provides an outline view of your project's contents. The contents of your project—files, folders, targets, executables, and other project information—are organized into groups. A **group** collects related files.

Using the Groups & Files list, you can:

- View your project's contents, organized hierarchically. You can choose how much of your project's contents to display at once.

- View the SCM status of files.

- Drag files, folders, groups, and other project items to rearrange and organize them.

- Rename files, folders, and other project items.

- Create additional Groups & Files list views to focus on multiple groups at once.

Group Types

The Groups & Files list contains two types of groups: *static groups* and *smart groups*, identified in Figure 1-2.

Figure 1-2   Groups & Files list
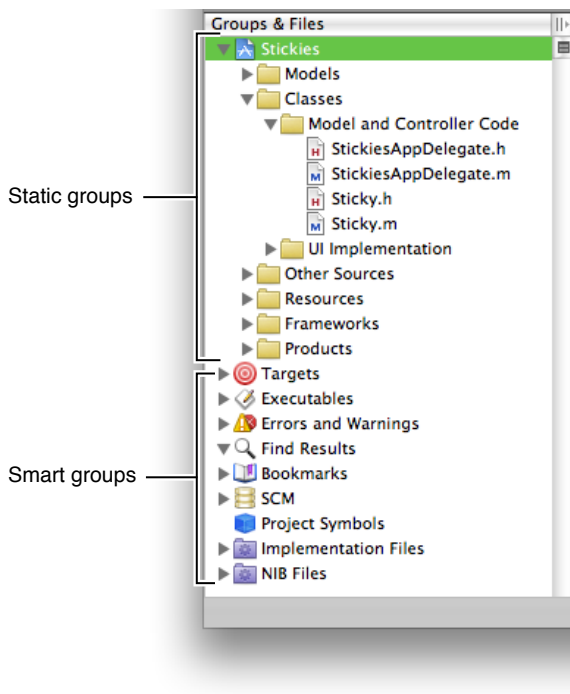
Static groups organize your project's source files, including header files, implementation files, frameworks, and other files. (Static groups are themselves grouped under the project group, which is named after the project and is represented by the blue project icon.) A static group, identified by a yellow folder icon, can contain any number of files and other static groups. Static groups help you organize the files in your project into manageable chunks. The project group is a static group that contains all the files, frameworks, libraries, and other resources included in your project.

Smart groups are subdivided into two types: built-in smart groups and custom smart groups.

- Built-in smart groups contain particular classes of components, files, symbols, or items. You cannot customize the contents of these groups. There are several built-in smart groups:

  - **Targets.** Contains the targets in your project. A **target** contains the instructions for creating a software component or product. Targets are described in more detail in Targets.

  - **Executables.** Contains all the executables defined in your project.

  - **Errors and Warnings.** Lists the errors and warnings generated when you build. This group is described further in "Viewing Errors and Warnings" in *Xcode Project Management Guide*.

  - **Find Results.** Contains the results of any searches you perform in your project. Each search creates an entry in this group. For more information on the Find Results group, see "Viewing Search Results" in *Xcode Project Management Guide*.

  - **Bookmarks.** Lists locations—files or specific locations within a file—to which you can return easily. For more information on the Bookmarks smart group, see "Defining Bookmarks" (page 36).

  - **SCM.** Lists all the files that have source control information. See "Managing Files Under Source Control" in *Xcode Source Management Guide* for details.

❑ **Project Symbols.** Lists the symbols defined in your project. This group is described further in "Viewing the Symbols in Your Project" in *Xcode Project Management Guide*.

■ Custom smart groups collect files that match a certain rule or pattern. These groups have purple folder icons and you can customize their contents using wildcard patterns or regular expressions. There are two types of smart groups: *simple filter* or *simple regular expression*. Xcode provides two predefined custom smart groups:

❑ **Implementation Files.** Contains the implementation files in your project, such as those with the extensions `c`, `cpp`, and `m`, to name a few.

❑ **NIB Files.** Contains the nib files used to create your product's user interface.

For more information on using static groups and smart groups to organize your project items, see "Grouping Files" (page 32).

Viewing the Contents of Groups

You have a couple of options for viewing the contents of a group in the Groups & Files list. If you prefer the outline view, you can open the group directly in the Groups & Files list. You can also select one or more groups to view their contents in a simple searchable list in the detail view (see "The Detail View" (page 16) for details).

To view or hide the contents of a group in the Groups & Files list, use its disclosure triangle.

To view the contents of a group in the detail view, select the group in the Groups & Files list. In general, the detail view shows more item information than the Groups & Files list.

You can display additional attributes for the items shown in the Groups & Files list using the shortcut menu in the list header (Control-click the header to show the shortcut menu). A column for each attribute appears on the left side of the list.
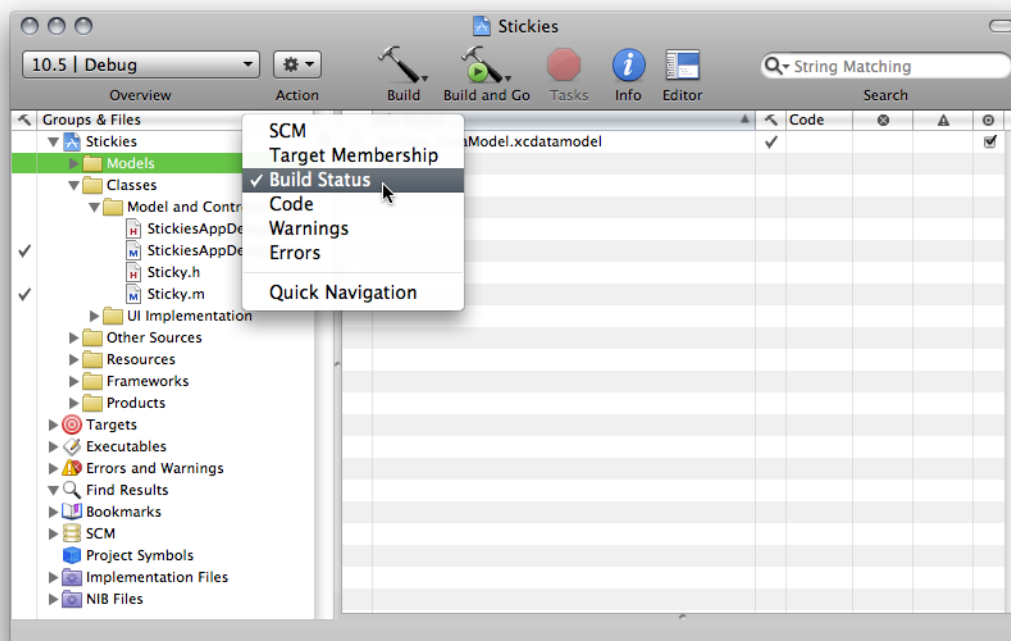
Displaying and Hiding Smart Groups

As described in "Group Types" (page 12), an Xcode project has a number of smart groups that organize particular types of project items. These groups help you find information such as symbols or build errors. However, you may not need to display all the smart groups in the Groups & Files list.

To specify which smart groups the Groups & Files list displays, use the Preferences submenu in the shortcut menu that appears when you Control-click an item in the Groups & Files list.

To rearrange a smart group, drag it to its new position in the Groups & Files list.

To delete a group, select it and choose Edit > Delete. You can restore deleted groups using the Preferences menu described earlier.

Splitting the Groups & Files View

In large projects, the Groups & Files list can get long, making it difficult to move items around. You can split the Groups & Files view for a project by clicking the Split button, shown in Figure 1-3.

Each Groups & Files view can display a different area of the Groups & Files list, making it easy to keep frequently accessed groups handy or to move items between groups. You can drag the resize control between the views to redistribute the space between them. To remove a Groups & Files view, click the Close Split button.

By default Xcode splits the view vertically. However, you can also split a view horizontally. To split a view horizontally, hold down the Option key while clicking the Split button.

Figure 1-3   Split Groups & Files view
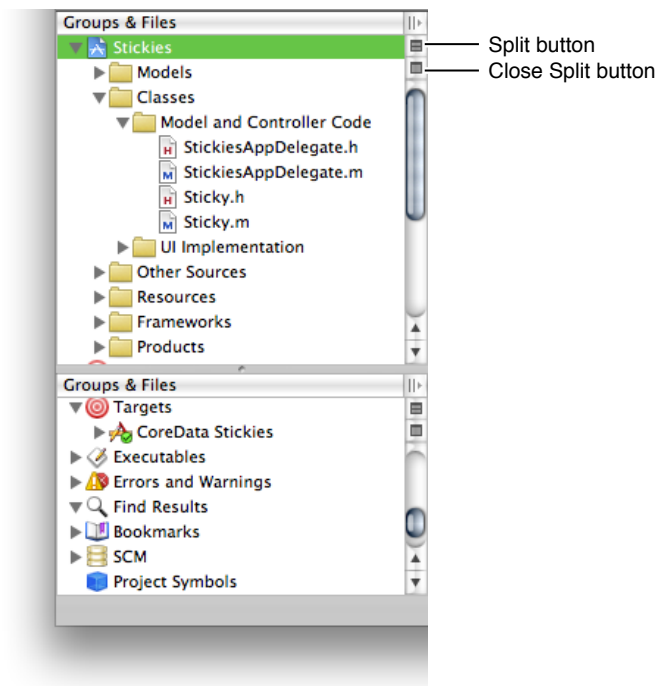
The Detail View

As you learned, the Groups & Files view lets you see the contents of your project in an organized outline. In contrast, the detail view shows you items in a flat list. You can quickly search and sort the items in this list, gaining rapid access to important information in your project.

You control the scope of the information shown in the detail view with your selection in the Groups & Files list. If the selected item is a group, the detail view displays information for all the members of that group and of any subgroups that it contains. You can select multiple items in the Groups & Files list; the detail view displays the selected items and their members.

> **Note:** The content of groups such as frameworks and bundles is shown in the detail view only when that framework or bundle is directly selected in the Groups & Files list, to avoid mixing items such as external framework headers and project headers.

Information Displayed in the Detail View

The type of information displayed in the detail view varies, depending on the item selected in the Groups & Files list. For example, if you select a group of source files in the Groups & Files list, the detail view displays each of the files in that group, along with information about those files, such as the file's build status or code size. However, build status and code size make no sense for errors and warnings, so when you select the Errors and Warnings group, you see a list of error and warning messages and the locations at which they occur.

Figure 1-4   Detail view columns
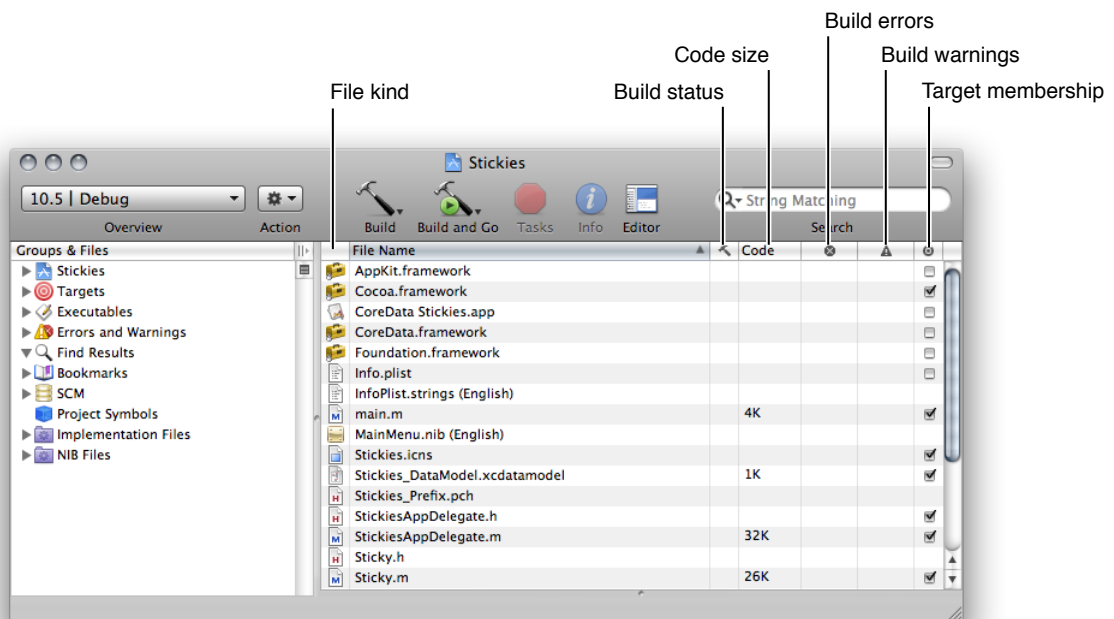
This is the information available in the detail view:

- **File kind.** The first column, with an empty column heading, shows an icon indicating the type of the file. For example, a nib file is marked by the Interface Builder file icon. The icon for a C++ class file displays the characters "C++".

- **File Name.** The File Name column displays the names of the files.

- **Build status.** The column marked by the hammer icon displays the build status of each file. If a file has been changed since the active target was last built, this column displays a checkmark, indicating that the file needs to be built. If the file is up to date, this column is empty.

- **Code.** The Code column displays the size of the compiled code generated from the file.

- **Build errors.** The column marked by the error icon displays the number of errors in the file. If this column is empty, the file either contains no errors or has not yet been built.

- **Warnings.** The column marked by the warning icon displays the number of warnings for the file. If this column is empty, the file either has no warnings or has not yet been built.

- **Target membership.** The column marked by the target icon indicates whether the file is included in the active target. If the checkbox next to a file is checked, then the active target includes that file.

- **SCM.** The SCM column shows the SCM status of the file.

- **Path.** The Path column shows the file system path to the item.

- **Comments.** The Comments column displays any note or other information that you have associated with the file in the Comments pane of the File Info window.

Not all of these columns are visible by default. You can choose which columns are shown in the detail view by using the shortcut menu that appears when you Control-click anywhere in the header. You can display many of these same columns in the Groups & Files list, using the same mechanism. For more information, see "The Groups & Files List" (page 12) and "The Detail View" (page 16).

> **Note:** Some columns in the detail view are required, depending on the currently selected group. These columns do not appear in the shortcut menu.

You can display the columns of the detail view in any order. To reorder the columns, drag the heading of any column to its new position.

You can also reveal the selected item in the detail view in the Groups & Files list. For example, if the current selection in the detail view is an individual source file, Xcode selects that file in the Groups & Files list, disclosing the contents of any static groups that the file belongs to, as necessary. To reveal the detail view selection in the Groups & Files list, choose View > Reveal in Group Tree.

Searching and Sorting in the Detail View

With the detail view you have a couple of ways to find and view information. You can sort the contents of the detail view according to the information in any of the visible categories simply by clicking the column heading for that category. For example, to sort by filename, click the File Name heading.

Using the search field in the toolbar, you can quickly search the contents of the detail view. As you type, Xcode filters the contents of the detail view, displaying only those items that have matching text in at least one of the columns.

The search field supports several types of search; you can choose the search type from the pop-up menu in the search field. Xcode supports the following searches:

- **String Matching.** Xcode determines a match using simple string comparison, filtering out items that do not match the string in the search field. This is the default type of search for the search field. It is also the fastest.

- **Wildcard Pattern.** Xcode uses the wildcard pattern in the search field to find items that contain the specified characters anywhere in any of the visible columns. For example, enter `*View*.h` to find all header files with "View" in their name.

- **Regular Expression.** Xcode uses the regular expression in the search field to find matching items. For example, to find all C implementation and header files in a source group, enter `\.(c|h)$`

Figure 1-5 shows an example of a sting matching search.

Figure 1-5   Searching for files with "delegate" in their name
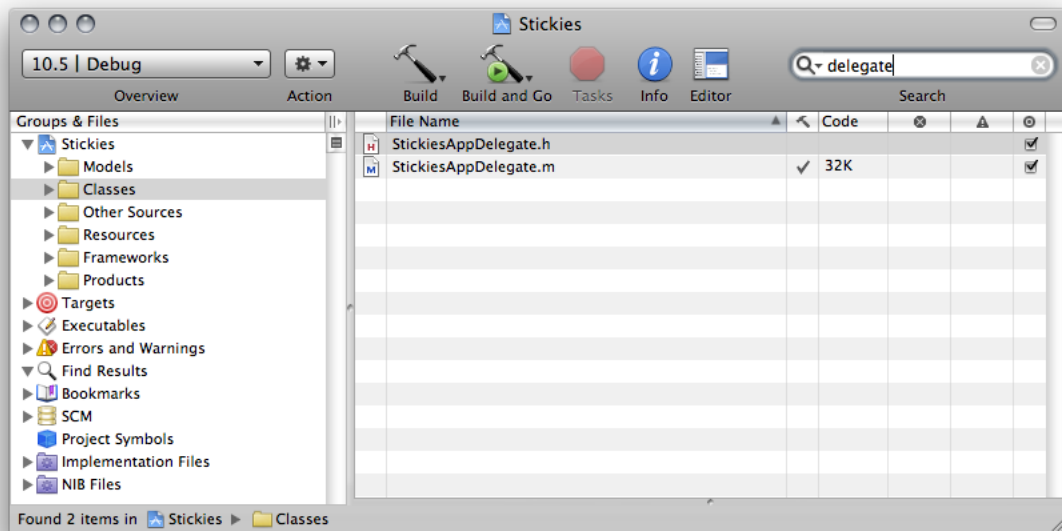
As you type, the status bar displays the scope of the search—the current selection in the Groups & Files list—and the number of items found. Pressing the Home key or choosing the project item (indicated by the project icon) from the search field's pop-up menu changes the focus of the search field to the whole project.

The Project Window Toolbar

The project window toolbar gives you quick access to the most common Xcode commands. Figure 1-6 shows the project window toolbar in the default layout (for more on project window layouts, see "Project Window Layouts" (page 20)).

Figure 1-6   Project window toolbar

- **Overview pop-up menu.** The Overview menu provides a set of build factors that specify how the to perform the next build process. This menu lets you specify the active SDK, active configuration, active executable, and active architecture. For detailed information about these factors, see "Setting Build Factors" in *Xcode Project Management Guide*.

- **Action pop-up menu.** The Action button lets you perform common operations on the currently selected item in the project window. The actions available from this menu are those appropriate for the selected item; they are the same actions available in the shortcut menu that appears when you Control-click the selected item. For example, when the current selection is a file, available operations include opening the file in a separate editor, performing source control operations, and grouping files.

- **Build and Go button.** The Build and Go button initiates common build actions, such as building, cleaning, and running. The triangle at the bottom of the button indicates that there are multiple actions associated with it. A single click on the button builds and runs your application. If you press and hold the mouse button over the button, you get a menu of actions.

- **Tasks button.** The Tasks button allows you to stop any Xcode operation currently in progress. The badge in the bottom-right corner of the Tasks button indicates the operation that is stopped when you click the button. If more than one operation is in progress, the Tasks button lets you select the one to stop from its pop-up menu. For example, if you have both a build and a search running, you can stop either operation by choosing it from the pop-up menu.

- **Info button.** The Info button brings up an Info window for the selected item or items. Info windows let you view and set various details of the selected item. See "The Inspector and Info Windows" (page 26) for more information.

- **Search field.** The search field allows you to search the items currently displayed in the detail view. As you type, Xcode filters the list of items in the detail view to include only those items with matching content in one of the visible columns. See "The Detail View" (page 16) for more information on using the search field to find items in the detail view.

You can customize the project window toolbar by choosing View > Customize Toolbar and dragging toolbar items into or out of the toolbar to get the set that is most useful to you.

The Project Window Status Bar

The project window **status bar** shows the progress of the current operation in Xcode. It gives you feedback during potentially lengthy tasks, such as building, and also displays the results of those tasks. In particular, the status bar lets you quickly access important information about project operations. From the status bar, you can:

- Click the progress indicator during an operation to open a more detailed account of the currently running operations in the Activity Viewer window, described in "Viewing the Progress of Tasks in Xcode" (page 27).

- Click the build result message, or error or warning icon, to open the Build Results window and view build system commands and output. For more information on the ways in which Xcode displays the status of build operations, see "Viewing Build Status" in *Xcode Project Management Guide*.

Project Window Layouts

There are many factors affecting the optimal workspace arrangement for you. Ask yourself questions such as the following: How much screen real estate do I have? What do I spend most of my time working on? How many projects do I normally have open at once?

Configuring your working environment to allow you to be as productive as possible is critical. Whatever your preferred workflow, Xcode provides several project window layouts for you to choose from. Xcode defines the following layouts:

- **Default.** This configuration provides the traditional Xcode project window arrangement, shown in Figure 1-1 (page 11). The default layout combines outline and detail views to let you quickly navigate your project.

■ **Condensed.** This layout provides a smaller, simpler project window with an outline view of your project contents and separate windows for common tasks, such as debugging and building.

■ **All-in-one.** This layout provides a single project window that lets you perform all the tasks typical of software development—such as debugging, viewing build results, searching, and so forth—within a single window.

You set the window layout for your environment in the General pane in Xcode preferences. From the Layout menu, choose the Default, Condensed, or All-In-One layouts. Selecting a layout from this menu shows a brief description of the layout below the menu. Note that you cannot change the project window layout when any projects are open; you must first close all open projects. The project window layout is a user-specific setting; it applies to all projects that you open on your computer.

This section describes each project window layout and the differences between them.

The Default Layout

The default project window layout, shown in Figure 1-1 (page 11), contains three main views:

■ **Groups & Files view.** Contains a list your project contents, as described in "The Groups & Files List" (page 12).

■ **Detail view.** Shows a flat list of the items selected in the Groups & Files list, as described in "The Detail View" (page 16). In the default layout, you can hide the detail view by collapsing the project window.

To hide the detail view, double-click the ![button] button in the Groups & Files view. Double-clicking the button a second time shows the detail view again.

In addition, you can customize the project window toolbar shown in each of these states. To do so, collapse or expand the project window to the appropriate state and customize the toolbar in the usual way, described in "The Project Window Toolbar" (page 19). Xcode stores the contents of the toolbar for each state separately.

■ **Text editor pane.** In the default layout, you can choose to view and edit all files within the project window, using the text editor pane. Or, you can choose to have Xcode use a separate editor window and use only the Groups & Files list and the detail view in the project window.

Although you can accomplish most of your daily development tasks in the project window, Xcode also provides a number of other task-specific windows that let you focus on a particular part of the development process. Table 1-1 shows the separate windows available in the default layout.

Table 1-1   Additional windows available with the default project window layout

Window	Use to
Build Results	View the build system output generated when you build a target.
Debugger	Debug your program; you can control execution of your code; view threads, stack frames and variables; and so forth.
SCM	View the status of files under source control.
Project Find	Search for text, symbol definitions, and regular expressions in your project.

Window	Use to
Console	View information or messages logged by your program when running in Xcode and interact with the debugger on the command line and see debugger commands and output.
Class Browser	View the class hierarchy of your project and browse classes and class members.
Breakpoints	View and edit all breakpoints set in your project.
Bookmarks	View your project's bookmarked locations in a dedicated window.

The Condensed Layout

The condensed layout provides a smaller, more compact version of the project window, shown in Figure 1-7. In this layout, the project window contains several panes, each showing a subset of the items in your project in the Groups & Files list. You can switch between these panes using the buttons below the toolbar.

Figure 1-7 The condensed project window
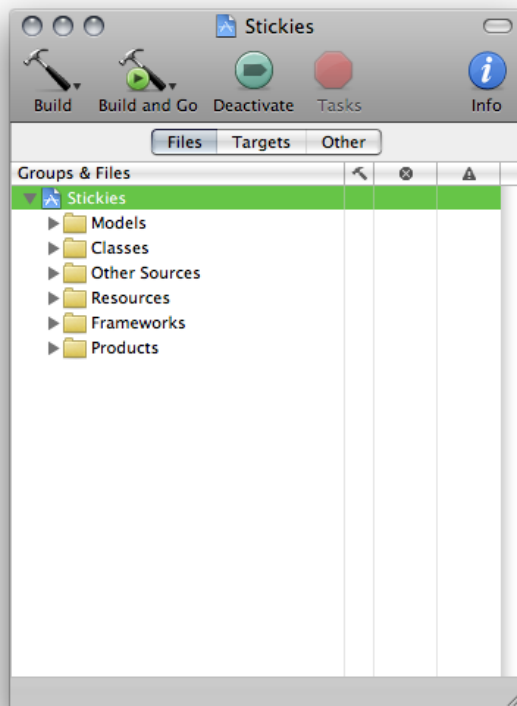
The condensed project window layout contains three panes:

■ **Files** shows your project and all the static groups in your project.

■ **Targets** shows the targets and executables defined in your project.

■ **Other** shows the remaining smart groups. This includes the standard smart groups defined by Xcode, as well as any smart groups you have added to the project.

You can jump to any of the built-in smart groups, opening the appropriate pane if necessary, by choosing an item from the Smart Groups submenu of the View menu.

The condensed layout provides the same additional windows as the Default layout, listed in Table 1-1 (page 21). The condensed layout also includes the additional windows shown in Table 1-2.

Table 1-2   Additional windows available with the condensed project window layout

Window	Use to
Editor	Edit project files. Although each of the available layouts let you open files in a separate editor window, the condensed project window is the only one that does not include an attached editor; when you open a file from the project window, Xcode opens a new editor window.
Detail	View and search your project's contents in a simple list. The condensed layout does not include a detail view in the project window; however, by choosing View > Detail, you can open a separate Detail window that includes a Groups & Files list on the left side of the window and a detail view on the right side.

The All-in-One Layout

The all-in-one project window layout provides a single project window in which you can perform all the tasks necessary for software development. In this layout, you can edit files, view project items in an outline view or detail view, view build system output, run and debug your executable, search, and more. The all-in-one layout provides two views, or **pages**. To switch between pages, use the Page toolbar item, shown in Figure 1-8 (page 24). The available pages are:

■ **Project.** This page lets you perform general project management tasks, such as searching, sorting and viewing SCM status. To display the project page, click the Project Page button.

■ **Debug.** This page includes an integrated debugger view, similar to the standalone debugger window available with the other layouts. To display the debug page, click the debug-page button.

In the all-in-one layout the project window has a page-specific toolbar, which contains items specific to the development tasks performed in that page.

The Project Page

The project page, shown in Figure 1-8, lets you perform typical project management tasks. You can view the contents of your project in outline view, search for project items in the detail view, perform a projectwide find, and view status for the files under source control in your project.

Figure 1-8   The all-in-one project window
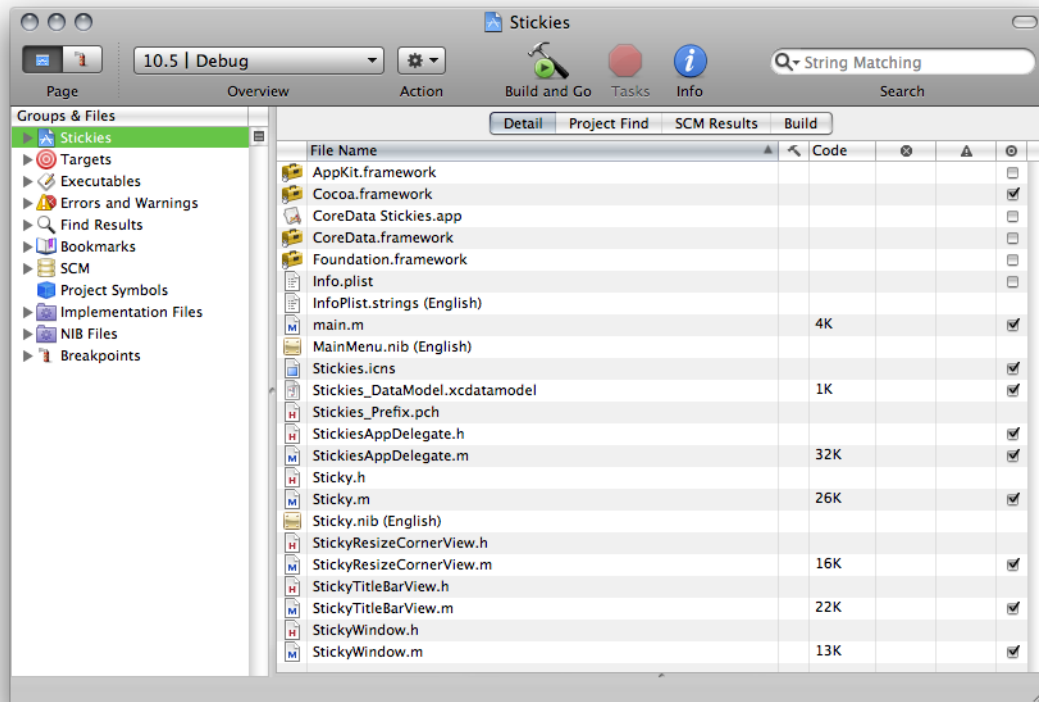
The project page contains the Groups & Files list, which shows the contents of your project in outline view; a text editor pane, which lets you edit source files right in the project window; and a tabbed view, which lets you switch between several panes, each of which provides an interface for a common project management task. These panes are:

■ **Detail.** This pane includes the detail view, which lets you view additional information about project items selected in the Group & Files list or quickly filter project items. The detail view is described in "The Detail View" (page 16).

■ **Project Find.** This pane lets you perform projectwide searches and view search results. The interface is the same as that provided by the Project Find window in other layouts.

■ **SCM Results.** This pane opens a dedicated view displaying only those project items under source control and their status. It is similar to what you see in the SCM window with other layouts.

■ **Build.** This pane provides an interface for common build tasks. It lets you see the progress of your build, view errors and warnings, and see the console log.

Other Windows in the All-In-One Layout

The all-in-one layout is designed to let you perform project management tasks in the project window; it does, however, include a few additional windows, listed in Table 1-3. These windows let you view content already available from the project window in a separate window, should you choose to do so.

Table 1-3   Additional windows available with the all-in-one layout

Window	Use to
Class Browser	View the class hierarchy of your project and browse classes and class members.
Breakpoints	View and edit all breakpoints set in your project.
Bookmarks	View your project's bookmarked locations in a dedicated window.
Project Find	Perform a projectwide search and view search results in a separate window. This window shows the same information as the Project Find pane of the project page.
SCM	View the status of files under source control in your project. This window contains the same information as the SCM pane of the project page.

Saving Changes to the Current Layout

The project window layouts give you the flexibility to choose the arrangement that best suits your preferred workflow. You can further customize your work environment by saving the changes that you make to the windows in an open project and applying them to all projects using that layout.

For example, in the condensed layout the project window has three panes, each of them focused on a particular subset of a project's groups. The Files pane shows only the project group, which contains all files and folders in the project. The Other pane shows all smart groups. If you want access to both your project files and your bookmarked locations in the same pane, you could add the Bookmarks smart group to the Files pane by dragging it from the Other pane to the Files button and then the Files pane.

> **Note:** If the Bookmarks smart group exists in another pane, choosing View > Smart Groups > Bookmarks opens that pane in the project window. However, if you have previously deleted the smart group from the project window, choosing View > Smart Groups > Bookmarks adds that smart group to the current pane.

To save this change, and have the Bookmarks smart group appear in the Files pane for all projects using the condensed layout, choose Window > Defaults.

In the dialog that Xcode displays, click Make Layout Default to save your changes to the current layout. Clicking Restore To Factory Default discards all of your changes—both current changes and those you've previously saved—to the current layout. Xcode restores the original configuration settings for the layout.

You can save changes to:

■ Window size and position

■ Visibility of views—whether they are hidden or revealed—in a window

■ Contents and visibility of Groups & Files panes (condensed layout)

■ The default set of toolbar items

The "Save window state" option in General preferences saves the state of the open windows for each project when you close that project. However, when you choose Window > Defaults, you save layout details that apply to all projects when you open them using the modified layout.

The Inspector and Info Windows

Knowing how to use Xcode's user interface to find and view project items and information is essential to working efficiently in Xcode. Xcode gives you a number of ways to find and access project contents. In previous sections, you learned how to use the Groups & Files list to see an organized outline view of your project and how to use the detail view to quickly filter project contents. Using these tools, you can view as wide or as narrow a cross-section of your project as you choose.

These tools, however, aren't as useful when you wish to view or modify individual items in your project. For this, Xcode provides inspector and Info windows. These windows are for managing the selected item's essential details. For any particular item in your project, its Info window and the inspector display the same information. However, the two windows behave differently.

The amount of information visible in the status bar is limited and it reflects only operations in the current project. To let you view a more detailed account of all operations in Xcode, Xcode provides the Activity Viewer window (see "Viewing the Progress of Tasks in Xcode" (page 27)).

The inspector tracks the current selection in the project window. As you select different files, targets, and groups in the project window, the inspector changes to show information about that item.

The Info window continues to display information about the items that were selected in the project window when you opened it, regardless of the current selection. You can have multiple Info windows open at a time, each describing a different set of items of your project.

The Info button in the project window toolbar displays an Info window for the currently selected item. You can add an Inspector button to the toolbar.

You can also use menu commands (or their keyboard shortcuts) to display Info windows or the inspector window. Choose File > Get Info to display an Info window. Hold down the Option key and choose File > Show Inspector to display the inspector.

Xcode provides Info windows and inspectors for the following project items:

- File and folder references
- User-defined smart groups
- Localized file variants
- Targets
- Executable environments
- Build phases
- The project itself

The type of information displayed depends on the type of item you are editing. Specific Info windows are described in more detail where the items that they modify—files, targets, projects, groups, and so forth—are discussed. In general, however, if you are at a loss as to how to make a change to a basic project item, try displaying its Info window. Throughout this document, wherever the inspector window is used, remember that you can also use an Info window, and vice versa.

Viewing the Progress of Tasks in Xcode

The Activity Viewer window lets you watch the tasks currently in progress in Xcode. While the Tasks button in the project window toolbar lets you control the progress of tasks in the current project (such as a program launched by Xcode in order to debug it), the Activity Viewer lets you see the progress of all Xcode tasks across all open projects. The Activity Viewer provides a single, persistent window that you can leave open to monitor the progress of all running tasks. To open the Activity Window, shown in Figure 1-9, choose Window > Activity Viewer.

You can also click the progress indicator in the project window status bar.

Figure 1-9   Activity Viewer window
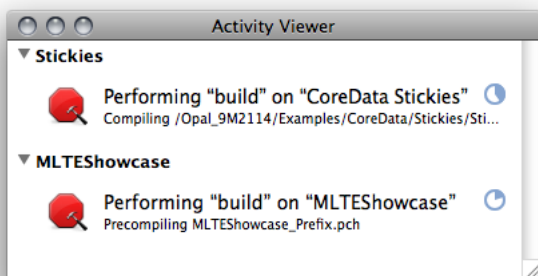
The operations in the Activity Viewer are grouped by project; you can show or hide the operations specific to any project using the disclosure triangle next to the project name. To stop any of the tasks shown in the Activity Viewer, simply click the stop sign icon next to that task. You can cancel the most recently initiated operation in the active project by pressing Command–period.

Project Organization

Some of the design decisions you make when you first set up a project—such as how many targets you need for your software development effort—affect your entire development experience. This chapter provides tips for partitioning and arranging the code and resources for a product as you develop with Xcode. It also describes some of the features Xcode provides that let you group information for rapid and easy access. For instance, you can save commonly accessed locations as bookmarks or in the favorites bar, or you can document project items by adding comments to them.

Software Development Tips

The following are some general guidelines for developing your software in Xcode. In subsequent sections, you'll get more detailed information to flesh out these tips.

- Follow standard software development practice to factor your software into logical units of reasonable size, which you can implement as applications, shared libraries, tools, plug-ins, and so on. In Xcode, each of these products requires one target.

- Put together projects and targets based on how you answer the following sorts of questions:

  - How do the products interact?

  - How many individuals (or teams) will be working on each product?

  - Do the products use different source control systems?

  - Must the products run on different versions of the target platform?

- For smaller development tasks, it generally makes sense to keep targets for related products in a single project.

- For products that are reasonably separate, and especially if they are to be implemented by separate teams, use multiple projects.

- Use cross-project references when targets in one project need to depend on targets in other projects.

- Use the information in "Build Locations" in *Xcode Project Management Guide* to ensure that a target can access the build products of other targets when needed.

Dividing Your Work into Projects and Targets

To develop software with Xcode, you are going to need at least one project, containing at least one target, and producing at least one product. Those are the basic structures you use for all your development.

Beyond that, there are no hard and fast rules for how you divide your work. For simple products or products that are closely tied together, you might use a single project with multiple targets. For large development tasks with many products, and especially with separate development teams, you'll want to use multiple projects and targets, perhaps connected with cross-project dependencies.

The following sections provide additional information and tips on laying out your software.

Identifying the Scope

In partitioning your software, many decisions depend on the scope of the design goal and the number of products it requires. For example, if you're working alone on a simple application, you can create a project for the desired application type (such as a Cocoa or Carbon application) and get right to work. Here are some of the decisions you might face:

■ If you decide to move some code to an internal library, you might add a target for the library.

■ To take advantage of existing code in another project, you might choose to develop the existing code as a shared library and add a dependency so that the application project depends on the shared library project and makes use of its output.

■ You might choose not to use a source code management (SCM) system, although even simple projects can sometimes benefit from such use.

Suppose, however, that you are working on a more complex software design, one that will be implemented by several individuals or development teams. Let's say you are asked to create an application for an easy-to-use recording studio. You may already have components of this application, such as a shared library for presenting music tracks. As you refine the product specification, you identify other common tasks that might belong in a shared library, tool, or companion application. You determine that the main application should rely on Apple technologies to provide certain features, such as burning CDs or making the application scriptable.

Eventually, you identify a set of products to build, which gives you a good idea of the scope of the task. And that in turn can help you determine how to partition it into Xcode projects and targets.

Trade-Offs of Putting Many Targets in One Project

You can help determine whether to put many or all your targets in one project by considering some of the trade-offs involved. Here are some of the advantages of combining multiple targets in one project:

■ Indexing works across all targets. Indexing information is required to access classes in the class browser, view symbols in the Project Symbols smart group, and to take full advantage of code completion. It is also necessary to use Command–double-click to jump to a definition and to use symbolic counterparts.

■ You can easily set up dependencies between targets in the project.

■ Anyone using the project has access to all its files.

■ If your computer has multiple CPUs or you have access to additional computers on your network, you can use parallel builds or distributed builds to improve the build time of a large project. See "Building in Parallel" in *Xcode Project Management Guide* and Reducing Build Times for details.

Here are some of the disadvantages of putting all your targets in one project:

- All of the code is visible to any individuals or teams using the project, even if they're working on only one target or a small subset of the overall project.

- The project size may become unwieldy; this can cause Xcode to take a long time to perform operations such as indexing.

- All targets must use the same SCM system.

- All targets must build using the same SDK.

- You can't use the Xcode debugger to debug two executables in one project at the same time.

Trade-Offs of Using Multiple Projects

There are also trade-offs in breaking up a software product into multiple projects and targets. Here are some of the advantages of using multiple projects:

- You can use the project as a unit for dividing work among different individuals or teams. The separate projects allow you to segregate the code (for example, to limit access to confidential information).

- Each project can be of a more manageable size, and Xcode can perform indexing, building, and other operations more quickly.

- If projects need to share outputs, they can build into a common directory, as described in "Build Locations" in *Xcode Project Management Guide*.

- You can use cross-project references to build other projects needed by the current project.

- Each project can use source code stored in a different SCM system. However, if individuals have physical access to other projects, it is still possible to look at SCM information from multiple projects that use the same SCM system.

- Each project can use different SDKs.

- Each project can define a separate list of build configurations.

- You can use the debugger to debug two or more executables at the same time, one in each project. This is useful when products communicate directly or otherwise interact.

Here are some of the disadvantages of having multiple projects and targets:

- You don't have cross-project indexing, and thus you have access only to symbols that are specifically exposed by other projects. This means, for example, that you can't automatically look up definitions in other projects unless you have physical access to them (not just to their end products).

  Similarly, you cannot take full advantage of other features that depend on indexing. That includes using code completion, using Command–double-click to jump to a definition, refactoring, and using symbolic counterparts.

- Management of many smaller projects is likely to incur additional overhead. For example, to set up a target that depends on other targets in multiple projects, you must first set up cross-project references.

  Similarly, use of multiple projects may require additional communication between teams.

- For an individual working on multiple projects, it may become unwieldy to switch between many open projects. This problem, however, may be alleviated by using the Organizer (see Using the Organizer for details).

- Individual projects are smaller and are less likely to be able to take advantage of distributed builds.

Grouping Files

Decisions about how to partition projects and their targets affect the design of your entire software development effort. It is also important that your work in a project be organized and accessible to you. Particularly in larger projects, the number of files can be daunting. To help arrange files into manageable chunks, Xcode lets you group them.

A **group** lets you collect related files together. A static group lets you group an arbitrary set of file, folder, and framework references in your project. A smart group, on the other hand, lets you group together files fitting a particular pattern or rule. This section shows you how to group files using static groups and smart groups.

Grouping Files into Static Groups

In the Groups & Files list, static groups look like folders, but don't have to correspond to folders on the file system. You can instead arrange files into groups that make sense to you while working on them in Xcode. For example, in a project containing multiple targets, your project could store all the nib files in one folder and all the implementation files in another folder on the file system. And in the Groups & Files list, you could group the files according to target; that is, the nib files and implementation files for target A would be in one group, the nib files and implementation files for target B would be in another group, and so on. A group doesn't affect how a target is built.

Creating a Static Group

You can create a static group in any of the following ways:

- Create an empty group. Choose Project > New Group and type the name for the group.

- Create a group from existing items. Select the items you want to group and choose Project > Group.

- Create a group based on an existing directory structure. Choose Project > Add to Project. Then select the folder, and select "Recursively create groups for added folders."

Adding Files to a Group

You can add files to a group at any time by dragging the file icons to the group's folder in the Groups & Files list. A line appears to indicate where you are moving the files. Xcode automatically expands groups as you drag items to them.

Deleting Groups

When you remove a group from your project, you can choose whether to remove the files in that group from the project or simply ungroup the files.

- To remove a group and the project's references to the files in that group, select the group, and press Delete.

- To remove a group and keep the files it contains, select the group and choose Ungroup from its shortcut menu.

Using Smart Groups

Smart groups are also represented by folder icons in the Groups & Files list; however, they are colored purple to distinguish them from static groups. As mentioned earlier, smart groups group files that adhere to a common pattern or rule. For example, you could define a smart group to collect all implementation files ending in `.m` or `.c`. When you make a change to your project that alters the set of files matching a smart group's rule—for example, adding a `.c` file—the smart group automatically updates to reflect the change. You do not need to add files to a smart group yourself. In fact, Xcode does not allow you to drag files to a smart group.

Creating a Smart Group

To create a smart group, choose Project > New Smart Group. Then choose one of the following options:

- **Simple Filter Smart Group.** Creates a smart group that collects files whose names contain a specified string.

- **Simple Regular Expression Smart Group.** Creates a smart group that uses a regular expression to specify the pattern that filenames must follow.

Xcode adds a smart group of the selected type to your project and brings up the Smart Group Info window, which allows you to configure the group. Xcode provides templates for each type of smart group; it uses these templates to configure new smart groups with a default set of options. For example, the default Simple Filter Smart Group collects all files with `*.nib` in their name. The default Simple Regular Expression Smart Group collects all C, C++, Objective-C, Objective-C++, Java, and AppleScript implementation files in your project.

Configuring a Smart Group

To configure a smart group, select the group in the Groups & Files list and open the Smart Group Info window, shown in Figure 2-1.

Figure 2-1   Smart Group Info window
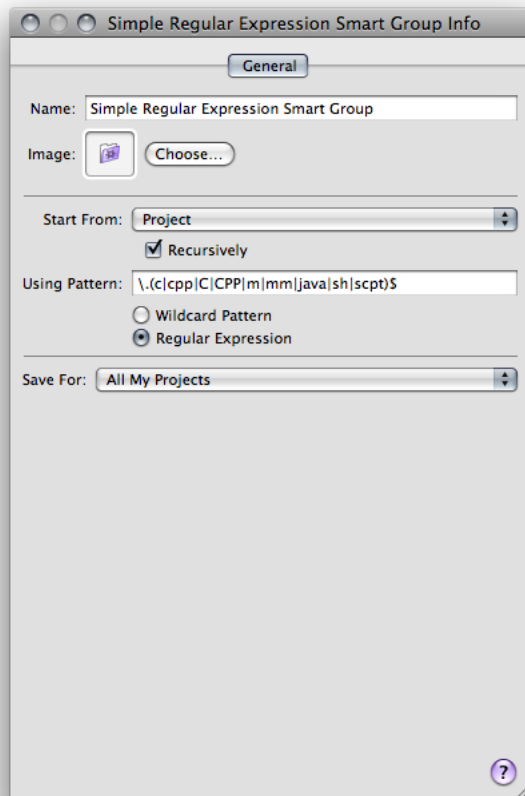
Here is what the Smart Group Info window contains:

■ **Name field.** The name of the smart group. By default, this is set to the type of the smart group, for example, "Simple Filter Smart Group."

■ **Image.** The icon used for the smart group in the project window. The default image for a smart group is the purple folder icon, but you can also use a custom image to represent your smart group. To change the icon, click the Choose button and navigate to the image file to use.

■ **Start From menu.** Specifies the directory from which the smart group starts its search for matching files; by default, this is the project folder. If the Recursively option is selected, the smart group also searches through subdirectories of that folder.

■ **Using Pattern field.** The actual pattern that files must match in order to be included in the smart group. As mentioned, this pattern can be either a simple string that the smart group filters on or a regular expression, depending on the setting of the options beneath the field. For examples of each of these, look at the smart group templates. The Simple Filter Smart Group template uses the pattern `*.nib`. Any files with `.nib` in their name are included in this smart group. The Simple Regular Expression Smart Group, on the other hand, uses the regular expression `\.(c|cpp|C|CPP|m|mm|java|sh|scpt)$` to collect all implementation files ending in any one of these suffixes, regardless of the case of the filename and extension. For more information on regular expressions, see the POSIX specification.

■ **Save For menu.** Determines the scope for which this smart group is defined. You can have the smart group appear in all your projects, or you can specify that the smart group be specific to the current project.

Viewing Groups and Files

As you learned in "The Project Window" (page 11), you have two ways to view the contents of your project. You can view the groups and files in your project in outline view to see a hierarchical representation of your project layout. Or you can view the groups and files in your project as a simple list, in the detail view. The detail view presents a flat list of all the files contained in the group selected in the Groups & Files list. For example, if you select your project in the Groups & Files list, the detail view shows all the files in the project. If you select an individual file in the Groups & Files list, only that file is displayed in the detail view. To view the contents of a group in the detail view, select that group in the Groups & Files list. See "The Detail View" (page 16) for more information about the detail view.

Saving Frequently Accessed Locations

In any project, there are locations that you find yourself accessing again and again: files that you edit frequently, headers that you browse, even smart groups that you use often. Xcode lets you save locations that you access frequently and provides ways for you to jump to these locations. This section describes how to take advantage of bookmarks and the favorites bar to provide quick access to project contents.

Adding Items to the Favorites Bar

The **favorites bar** in the project window lets you save frequently accessed items and return to them quickly. To show the favorites bar, choose View > Layout > Show Favorites Bar.

To add an item to the favorites bar, simply drag it to the favorites bar in the project window. You can save any of the same locations you can bookmark, including files, documentation, URLs, and so forth. In addition, you can add smart groups and static groups to the favorites bar. The Groups & Files list can get quite long as you reveal the contents of more and more groups, scrolling the items at the bottom of the list out of view. You can add groups—including any of the built-in smart groups—to the favorites bar to quickly jump to that group in the Groups & Files list.

To reorder items in the favorites bar, drag them to the desired location; dragging an item off of the favorites bar removes the item from the favorites bar. To rename an item in the favorites bar, Option-click the item and type the new name. This changes the name of the entry in the favorites bar; it does not affect the name of the actual item that the entry represents.

To open a saved location, click it in the favorites bar. If the item in the favorites bar is a container, such as a group or folder, it is a pop-up menu that you use to navigate through the contents. Each of the items in the favorites bar serves as a proxy for the actual item. Thus, Control-clicking the item brings up the item shortcut menu, which allows you to perform operations appropriate for the selected item.

Defining Bookmarks

Another way that Xcode lets you navigate your project and provide easy access to the information you need is with bookmarks. If you have files or locations in your project that you access often, you can bookmark them and return to those locations at any time simply by opening the bookmark.

To create a bookmark, select the location you want to bookmark and choose Edit > Add to Bookmarks.

Xcode automatically names some bookmarks. For items such as symbols, Xcode prompts you for the name of the bookmark; you can enter a name to help you remember which location the bookmark indicates, or you can use the name Xcode assigns. You can bookmark project files, documentation, URLs, and so forth.

You can see the bookmarks in your project in several ways:

- **Bookmarks smart group.** Select the Bookmarks group in the Groups & Files list to see the bookmarks in the detail view. The detail view shows the name and file of all the bookmarks in your project. If the editor pane is open, selecting a bookmark opens that location in the editor. Otherwise, you can double-click the bookmark to open the bookmarked location in a separate editor window.

- **Bookmarks window.** To open this window, double-click the Bookmarks smart group. You can see all bookmarks in your project in this window; double-click any of these bookmarks to open the location.

- **Bookmarks menu.** This pop-up menu resides in the navigation bar of the text editor. It contains the bookmarks for the current file. Choose any bookmarked location from this menu to open it in the editor, as described in "A Tour of the Text Editor" (page 51).

- **Documentation window.** The Bookmarks list in the left side of the Documentation window contains documentation bookmarks.

Adding Comments to Project Items

To help you keep track of your project contents, you can write comments and associate them with any of the items in your project. Xcode remembers these comments across sessions. In this way, you can document your project and its components, keep design notes, or track improvements you still wish to make. This is especially useful if you are working with large or complex projects that have many pieces, or if you are working with a team of developers and have multiple people modifying the project.

For example, imagine a large project containing targets that build a suite of applications, a framework used by each of those applications, a command-line tool for debugging, and a handful of plug-ins for the applications. With such a large number of targets it might be hard to keep track of exactly what each of the products created by those targets does. To make it easier to remember what each of them does, you could add a short description of the product that it creates to the target's comments. This also aids others who may be working on the project with you; if they can read the comments, they can quickly get up to speed with the project and easily learn about changes made by other members of the development team.

To add comments to a project item:

1. Select that item in the Groups & Files list or in the detail view and open its Info window.

2. Click Comments to open the Comments pane.

This pane contains a single text field into which you can type any information you wish. To add comments, click in the Comments field and type your entry. You can also link to additional resources from the Comments pane; hypertext links, email addresses and other URLs are actionable in this field.

You can add comments to any project item except smart groups, including projects, targets, files, and executables. You can view comments you have added to project items in the detail view and you can search the content of those comments using the search field. If the Comments column is not already visible, use the shortcut menu of the detail view header to display it.

General Preferences

The General pane of Xcode preferences lets you control general environment settings for the Xcode application, such as your project window configuration and windowing preferences. Figure 2-2 shows the General pane.

Figure 2-2   General preferences pane
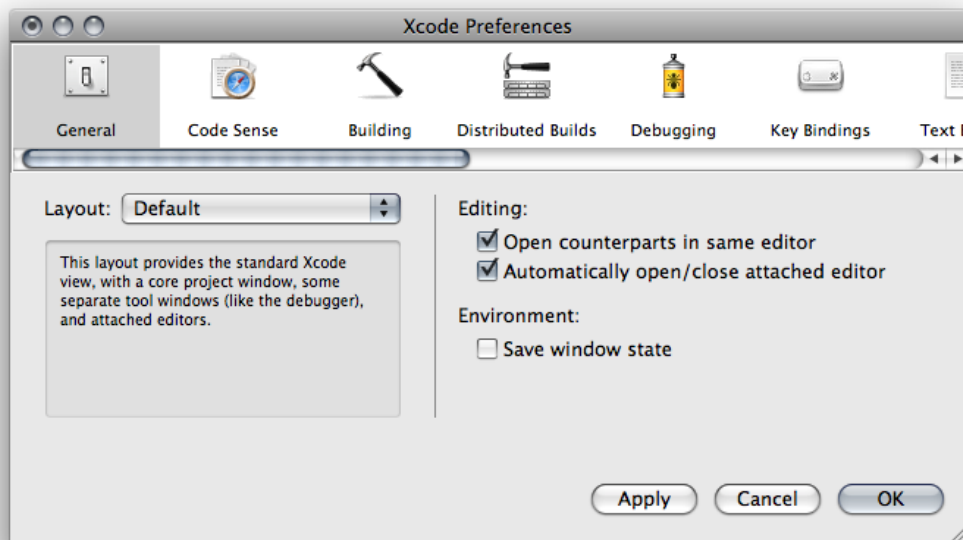
Here is what the pane contains:

- **Layout.** This menu lets you choose the project window configuration for all projects. See "Project Window Layouts" (page 20) for a description of the available layouts.

- **Editing.** These options control Xcode's windowing policy for editor windows. See "A Tour of the Text Editor" (page 51) to learn more about the Xcode editor. The options are:

  - **Open counterparts in same editor.** Specifies how Xcode displays files when jumping to a related header or source file, or to a related symbol definition or declaration. If this option is selected, Xcode opens file and symbol counterparts in the current editor window; otherwise, it opens a separate editor to display the counterpart. "Opening Header Files and Other Related Files" (page 39) describes how to jump to a file's or symbol's counterpart.

❏ **Automatically open/close attached editor.** Specifies when Xcode shows or hides the editor pane attached to the project, Build Results, Project Find, or Debugger windows. If this option is selected, Xcode automatically shows the attached editor for a window when you select an editable item. See "Using Text Editor Panes" (page 57) for more information on attached editors.

■ **Environment.**

❏ **Save window state.** Specifies whether Xcode saves the state of the windows in a project across sessions. If this option is selected, Xcode saves the state of a project's open windows when you close the project and restores those windows when you next open the project. Otherwise, Xcode does not save the state of a project's windows; reopening a project opens only the project window.

File Management

For each file, framework, and folder added to a project, Xcode stores a number of important details, such as the path to the file and the file's type. You can view and edit information for file, framework, and folder references in the File Info window.

This chapter describes how to edit file, folder, and framework references in your project. It also describes how to change the way in which Xcode handles a file by changing its type and how to control the way a file is displayed and saved by changing the file encoding and line ending.

Opening Files

Xcode offers several ways to open files, depending on your current context. The following sections describe these mechanisms.

Opening Project Files

If you already have a project window open, you can open a file by selecting it in the Groups & Files list in that window. If you have the text editor pane open inside the project window, clicking the filename opens the file in the editor. Otherwise, double-clicking the file in the Groups & Files list opens the file in a separate editor window.

If a filename is in red, Xcode cannot find the file. Set the correct path to the file in the File Info window.

If you have the text editor open and you have previously opened the file you want to open, you can choose the file from the File History pop-up menu in the navigation bar (see for more information).

Opening Header Files and Other Related Files

You can quickly open a header or source file that's related to the file displayed in the text editor.

To open the related header for an implementation file open in the editor, and vice versa, choose View > Switch to Header/Source File.

For example, if `main.c` is in the editor, this command opens `main.h`; if `main.h` is in the editor, it opens `main.c`.

You can also view the current file's include group (all the files that the file includes, as well as all the files that include it). To view the list of files that the file in the editor includes and that include this file, use the Included Files pop-up menu, , in the navigation bar of the text editor. The name of the current file is in the middle of the menu. Above it are the names of the files that this file includes. Below it are the names of the files that include this file. To open one of the files, choose it from the menu.

Opening Superclasses and Subclasses

Similar to opening header files, you can open the declarations of superclasses and subclasses in C and Objective-C source files using the Class Hierarchy pop-up menu, , in the navigation bar.

Opening Files by Filename or Symbol Name

There may times when you need to open a file whose location you don't know or want to open the file that defines a particular symbol, such as a variable, method, or class. To assist you in those occasions, Xcode provides the **Open Quickly** command, which lets you open files by filename or by the symbols they define. Open Quickly searches are case insensitive and limited to the current project and Active SDK (see "Building with Xcode" in *Xcode Project Management Guide* for more information).

To open a file quickly:

1. Choose File > Open Quickly to display the Open Quickly dialog (Figure 3-1).

2. In the search field, type the name of the file or symbol you want to view.

   You can also select a filename or symbol name in the text editor and display the Open Quickly dialog to prefill the dialog's search field with the selected text. When opening header files referenced in `#include` and `#import` statements, you need only to place the cursor on the referent line and display the Open Quickly dialog to start the header-file search.

3. From the search results list, select the files you want to open and click Open.

Figure 3-1   The Open Quickly dialog
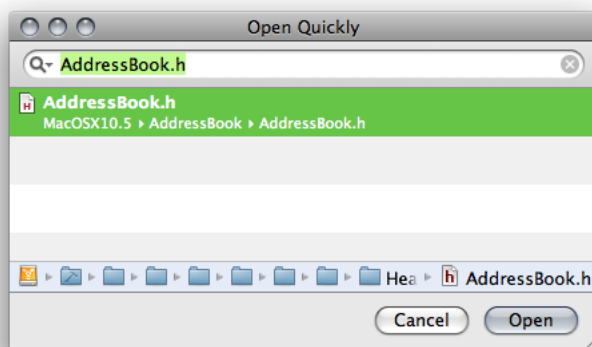
As you type text in the search field, Xcode updates the search results list. The search field also maintains a list of previously used search terms.

In filename-based searches, Xcode searches for header files, implementation files, model files, nib files, plist files, and project packages. In symbol name–based searches, it searches source-code files only.

Specifying How Files Are Opened

You can temporarily change how a file is viewed and edited, or permanently change how files of a certain type are viewed and edited. For example, you can choose to view a particular HTML file as plain text, so you can edit it instead of viewing it as rendered HTML. To specify how a file is viewed, you can do any of the following:

- To have files of a certain type always open in a specific editor, change the preferred editor for that file type to that editor. (See "Specifying the Editor Type for a File Type" (page 41).)

- To have files of a certain type always open in the application specified for them in the Finder, change the preferred editor for that file type to Open With Finder. (See "Opening Files with Your Preferred Application" (page 44).)

- To temporarily force Xcode to treat a file as a different file type, and open it with the appropriate editor, use the Open As command in the file's shortcut menu (Control-click).

- To temporarily force Xcode to open a file with the default application chosen for it in the Finder, use the Open With Finder command in the file's shortcut menu.

> **Note:** HTML files are handled differently. If Xcode determines that an HTML file is documentation, Xcode assumes you want to view the file and displays the file with its built-in HTML viewer. Otherwise, Xcode assumes you want to edit the file and uses its built-in text editor.

Specifying the Editor Type for a File Type

You can permanently change how Xcode edits a particular type of file. In particular, you can specify how files of a certain type are treated, and you can choose which editor Xcode uses to handle those files. For example, you can choose to view all HTML files as plain text so that you can edit them. Or you can choose to edit all your source files in BBEdit.

These are the main editor types in Xcode:

- **Source Code File.** The Xcode text editor lets you view and edit these files.

- **Plain Text File.** The Xcode text editor lets you view and edit these files.

- **HTML File.** The text Xcode editor allows you only to view these files.

- **External Editor.** These files are opened with an external text editor. See "Opening Files with an External Editor" (page 43) for details.

- **Open With Finder.** These files are opened with the application the Finder associates with them. See "Opening Files with Your Preferred Application" (page 44) for details.

File Types preferences lists all the folder and file types that Xcode handles and the editor type for each of those types. These file and folder types are organized into groups, from the most general to the most specific. The Preferred Editor column shows the editor type to which each file type is associated.

To change the editor type for a particular file type, choose the editor type from the Preferred Editor pop-up menu of the file type. For example, to make Xcode let you edit HTML files—except documentation—in the text editor as source code, choose Source Code File from the Preferred Editor pop-up menu for the `file/text/text.html` file type, as shown in Figure 3-2.

Figure 3-2   File Types preferences pane
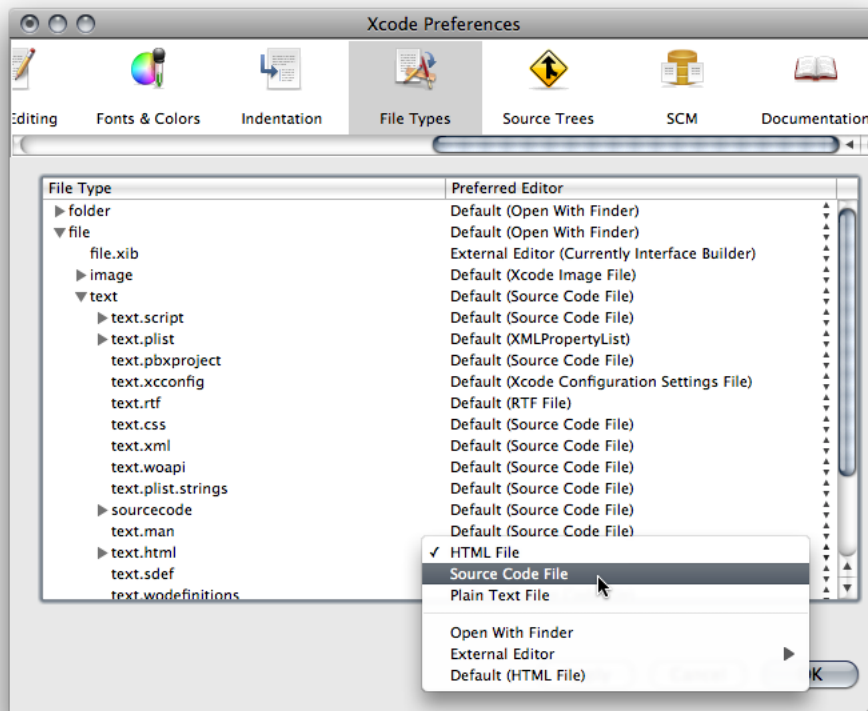
> **Note:** You must close and reopen your project for the changes to take effect.

File type entries are organized into groups, from most general to most specific. For example, the `audio.mp3` and `audio.aiff` file types belong to the "audio" group, which belongs to the "file" group. In this way, you can control the default editor used for an entire class of files. To see the file types in a group, click the disclosure triangle next to that group.

You can also specify an external editor to use or have Xcode use the user's preferred application, as specified by the Finder, when opening files of a given type, as described in "Opening Files with an External Editor" (page 43) and "Opening Files with Your Preferred Application" (page 44).

Opening Files with an External Editor

Xcode does not limit you to using its text editor to view and edit your files. You can specify an external editor as the preferred editor for opening files of a given type. To choose an external editor for all files of a particular type:

1. Open Xcode preferences and go to the File Types pane.

2. Find the appropriate file type and click in the Preferred Editor column; a menu appears.

3. Choose an option from the External Editor submenu.

    These are the supported external editors:

    - BBEdit

    - Text Wrangler

    - SubEthaEdit

    - `emacs`

    - `xemacs`

    - `vi`

> **Note:** Some of these external editors do not appear in the External Editor menu unless they are installed on your computer.
>
> Support for `vi` in Xcode is limited to opening the file in the editor.

For example, to edit all your source files with BBEdit, choose External Editor > BBEdit from the Preferred Editor pop-up menu for the `file/text/sourcecode` file type.

There are some restrictions when using an external editor:

- When you build a project, Xcode lists modified files and asks whether you want to save them. Files in BBEdit and Text Wrangler are listed, but files in other editors are not. You need to save those files yourself before starting a build.

- When you double-click a find result or a build error, most editors do not scroll to the line with the find result or error. BBEdit and Text Wrangler can.

To use `emacs` as an external editor, you must add these lines to your `~/.emacs` file:

```
(autoload 'gnuserv-start "gnuserv-compat"
        "Allow this Emacs process to be a server for client processes." t)
(gnuserv-start)
```

Xcode defines the `PBXEmacsPath` and `PBXXEmacsPath` user defaults for specifying paths to the `emacs` and `xemacs` editors, respectively. By default, these paths are set to `/usr/bin/emacs` and `/usr/bin/xemacs`; however, you can change them to use a custom built editor. For more information, see *Xcode User Default Reference*.

Opening Files with Your Preferred Application

You can choose to open a file with the application chosen for it in the Finder. This lets you open files that Xcode cannot handle, or view a file using your preferred file editor. If you edit a file in almost any other application, Xcode cannot save it for you before building a target. Some applications, such as Interface Builder and BBEdit, communicate with Xcode and so can save your files before your project is built. Check the application's documentation to find out whether it can, too.

To always have Xcode use your preferred application to open files of a certain type, find the appropriate file type in File Types preferences and choose Open With Finder from its Preferred Editor menu.

Note that you can set this preference only for file types that Xcode recognizes. To open files that Xcode cannot handle, or to temporarily override the settings in the File Types preferences pane and open a file using the Finder-specified application: In the Groups & Files list or detail view, choose Open With Finder from the file's shortcut menu.

If you have the text editor pane of a window (such as the project window) open, selecting a file still loads the file in the editor. But if you configure Xcode to use an external editor to edit the file, you cannot edit the file within Xcode. That is, the file is read-only in the Xcode text editor.

Saving Files

Xcode indicates which files you've modified by highlighting their icons in gray in the Groups & Files list, the detail view, and the File History pop-up menu. You can save your changes in a number of ways:

- To save changes to the current file, choose File > Save.

- To save a copy of a file, choose File > Save As. Xcode saves a copy of the file under the name you specify. If the file is part of your project, Xcode also changes the file reference in your project to refer to the copy.

- To make a backup of a file, hold the Option key and choose File > Save a Copy As. Xcode saves a copy of the file under the name you specify, but does not modify the file reference in your project, if one exists.

- To save all open files, hold Option and choose File > Save All.

Xcode can also be configured to automatically save all changed files before beginning a build. To specify whether files are saved automatically when you build a target:

1. Choose Xcode > Preferences and click Building.

2. Use the For Unsaved Files menu, to choose the behavior you want.

If you don't have write permission for a file, Xcode warns you when you try to edit it. You can choose to edit such files, but you can save your changes only if you have write permission for the containing folder. In this case, you can choose whether Xcode changes the file's permissions to make it writable.

To have Xcode change the file's permissions, select the "Save files as writable" option in Text Editing preferences. Otherwise, Xcode preserves the file's current permissions.

Closing Files

Files in a project remain open until you explicitly close the file or close the project. To close the current file in a text editor, choose File > Close File.

Deleting Files

To remove a file from a project and, optionally, to delete it from your file system, select the file in the Groups & Files list or the detail view and choose Edit > Delete.

Viewing File Information

You can view and edit settings for files, frameworks, and directories using the File Info window. To display the File Info window, select the item to edit and choose File > Get Info.

> **Note:** When dealing with items in static groups, you're really manipulating **file references**. These references point to files, in your file system. For more information on file references, see "Files in Xcode" in *Xcode Project Management Guide*.

If you select more than one file, Xcode displays one File Info window that applies to all the selected files. Attributes whose values are not the same for all selected files are dimmed. Changing an attribute's value applies that change to all selected files.

The File Info window for a file, folder, framework, or static group contains the following panes, shown in Figure 3-3:

- **General.** Lets you modify a number of main file attributes, such as filename, location, reference style, and character encoding.

- **Targets.** Lets you view and change which targets include the file.

- **Build.** Lets you specify additional compiler flags for the file. The Build pane appears only when the file being inspected is a source code file.

- **SCM.** Lets you view SCM information for the file. This pane is available only for files under source control.

- **Comments.** Lets you add notes, documentation, or other information about the file. See "Adding Comments to Project Items" (page 36) for more information on adding comments.

Figure 3-3   The File Info window
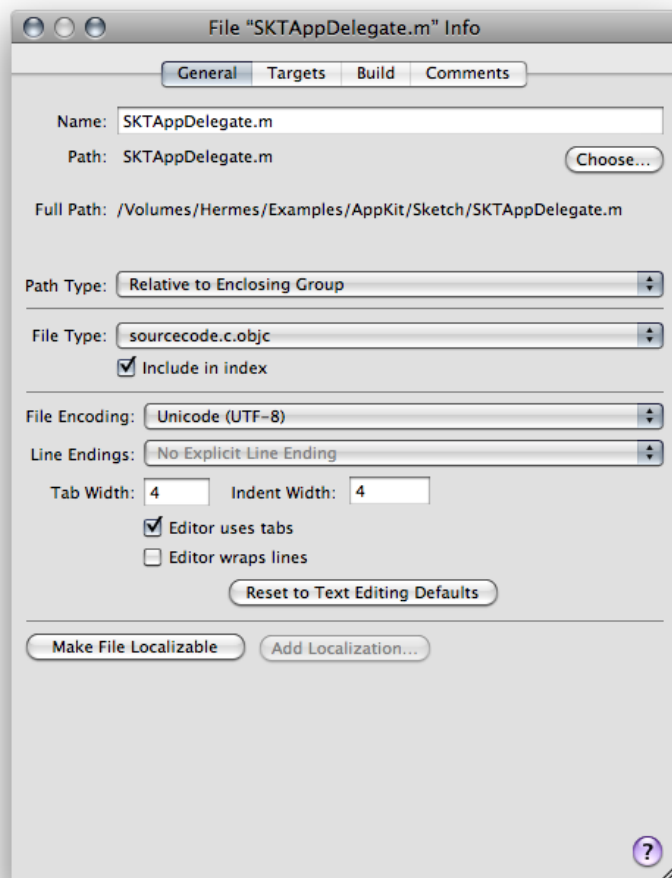
Here is what you can see and edit in the General pane of the File Info window:

- **Name field.** Displays the file's name. To rename the file, type the new name in this field.

- **Path field.** Shows the location of the file; that is, it shows the path to the file. To change this location, click the Choose button next to the path. You get a dialog that lets you choose a new path.

- **Path Type pop-up menu.** Indicates the reference style used for the file. These reference styles are described in "How Files Are Referenced" in *Xcode Project Management Guide*. To change the way the file is referenced, choose a style from this menu.

- **File Type pop-up menu.** Lets you explicitly set the file's type, overriding the actual file type of the file. How to change a file's type is described in "Overriding a File's Type" (page 48).

- **"Include in index" checkbox.** Controls whether Xcode includes the file when it creates the project's symbolic index. See "Source Code Indexing" in *Xcode Overview* for more information.

- **File Encoding pop-up menu.** Specifies the character set used to save and display the file. File encodings are discussed further in "Choosing File Encodings" (page 47).

- **Line Endings pop-up menu.** Specifies the type of line ending used in the file. Line endings are discussed in "Changing Line Endings" (page 48).

- The next several options control tab and indent settings for the individual file. The Tab Width and Indent Width fields control the number of spaces that Xcode inserts when it indents your code or when you press the Tab key when you edit the file in the text editor. To change either of these values, type directly in the field.

  "Editor uses tabs" option. Specifies whether pressing the Tab key inserts spaces or a tab when you are editing this file in the text editor. Controlling indentation in the editor is discussed further in "Tab and Indent Layout Options" (page 60).

  "Editor wraps lines" option. Specifies whether the text editor wraps lines that are too long to fit in the window.

  Reset to Text Editing Defaults button. Restores the line ending, file encoding, tab and indent settings to the built-in defaults.

- **Make File Localizable and Add Localization buttons.** Let you customize files for different regions.

Choosing File Encodings

The file encoding of a file defines the character set that Xcode uses to display and save a file. If you type a character that isn't in the file's encoding, Xcode asks whether you want to change the encoding. Xcode uses the default single-byte string encoding or Unicode if the file contains double-byte characters.

To change the file encoding for one or more files:

1. Select the files and open a File Info window.

2. In the General pane, choose the desired file encoding from the File Encoding pop-up menu.

   Generally Unicode (UTF-8) is best for source files and Unicode (UTF-16) is best for `.strings` files.

When Xcode next saves the file, it uses the new file encoding.

To choose the default file encoding for new files, use the Default File Encoding pop-up menu in Text Editing preferences.

> **Note:** GCC, the compiler used by Xcode for C-based languages (such as C, Objective-C, and C++), expects its source files to contain only ASCII characters, with the exception that comments and strings can contain any characters. Also, some encodings use escape sequences to handle non-ASCII characters, and those escape sequences can cause unexpected results when GCC interprets them as ASCII characters. For example, some characters in the Japanese (Shift JIS) encoding look like `*/` and will end your comment before you intended. Unicode (UTF-8) avoids this confusion.

Changing Line Endings

UNIX, Windows, and Mac OS X use different characters to denote the end of a line in a text file. Xcode can open text files that use any of these line endings. By default, it preserves line endings when it saves text files. However, other utilities and text editors may require that a text file use specific line-ending characters. You can change the type of line endings that Xcode uses for a single file, or you can change the default line ending style that Xcode uses for new or existing files.

To change an individual file's line endings, select the file in the project window and open its File Info window. In the General pane of the info window, use the Line Endings pop-up menu to choose Unix Line Endings (LF), Classic Mac Line Endings (CR), or Windows Line Endings (CRLF).

To choose the default line endings used for all files, display the Text Editing preferences pane, and choose Unix (LF), Mac (CR), or Windows (CRLF) from the Line Encodings pop-up menus:

- **For new files.** Lets you choose the default line encoding that Xcode uses for all new files.
- **For existing files.** Specifies the type of line encoding that Xcode uses when saving existing files that you have opened for editing in Xcode. To have Xcode preserve line endings for existing files, choose Preserve from this menu; this is the default value for the menu.

Generally, you don't need to worry about line endings. If you find that you must change line endings from the defaults assigned by Xcode, keep these guidelines in mind when deciding which line endings to use:

- Most Mac OS X development applications, including CodeWarrior and BBEdit, can handle files that use UNIX, Mac OS, or Windows line endings.
- Many UNIX command-line utilities, such as `grep` and `awk`, can handle only files with UNIX line endings.

Overriding a File's Type

By default, Xcode uses the type stored for a file in the file system to determine how to handle that file. A file's type affects which text editor Xcode opens the file in (the internal editor or an external one), how the file is processed when you build a target that includes the file, and how Xcode formats the file when syntax formating is turned on. You can change the way that Xcode handles a file by overriding the file's type.

The File Type pop-up menu in the General pane of the File Info window lets you explicitly set the file's type, overriding the actual file type of the file. The File Type menu lists all the file types that Xcode is aware of; to set a file's type, choose it from this menu. Choosing Default For File discards any explicit file type set for the file in Xcode and reverts to using the type stored for the file in the file system.

For more information on how Xcode determines how to process files of a certain type, see "Build Rules" in *Xcode Build System Guide* and "Adding Files to a Build Phase" in *Xcode Build System Guide*. For more information on how Xcode chooses the editor to use for files of a certain type, see "Specifying How Files Are Opened" (page 41).

The Text Editor

Xcode contains a full-featured text editor for editing your project's text files. You have many options for using this editor to view and modify the text files in your project; you can edit files in a dedicated editor window or use the editor pane attached to most Xcode windows. You can also choose whether to have multiple editor windows open at once or use a single editor window for all the text files that you open.

This chapter describes the Xcode text editor, shows how to open files in a standalone window or in an editor pane, and how to control the appearance of the editor.

A Tour of the Text Editor

You can view the text editor in two ways:

- **A text editor window:** A window whose main purpose is to let you edit a file.
- **A text editor pane:** Also known as an attached editor, these panes are part of other windows, such as the project window, debugger window, or build results window.

In either case, the content looks the same and the same controls are available. When you open a file in the text editor, you see something similar to Figure 4-1.

Figure 4-1   The text editor view
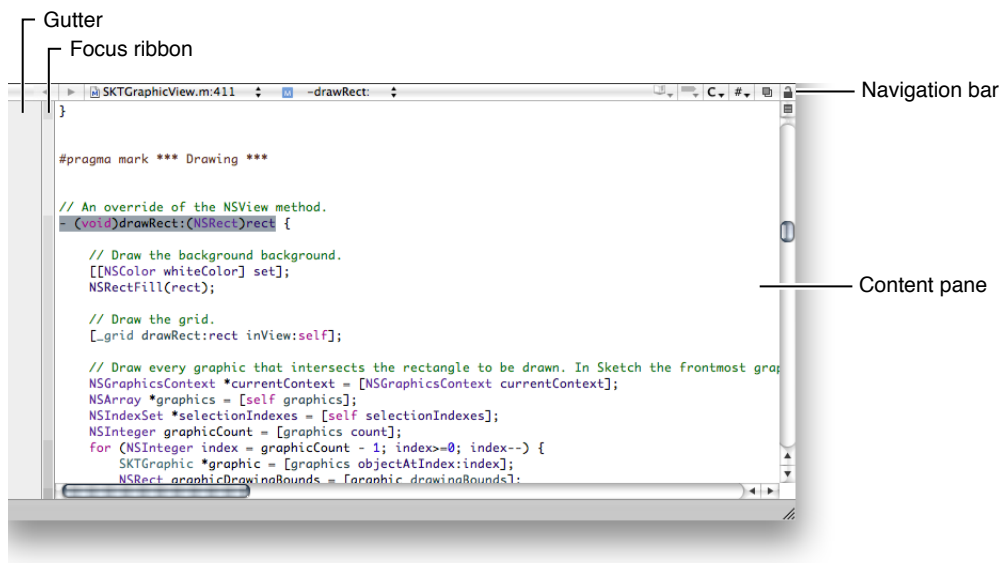
Here's what the text editor contains:

■ **Gutter.** The gutter displays file line numbers, as well as information about the location of breakpoints, errors, or warnings in the file. See "Displaying the Gutter" (page 77) to learn more about the contents of the gutter, as well as how to show and hide the gutter.

■ **Focus ribbon.** The focus ribbon allows you to navigate the scope of a code file and to fold and unfold parts of a code file. To learn more about folding/unfolding code, see "Code Folding" (page 66).

■ **Navigator bar.** The bar along the top of the editor contains several menus and buttons that let you quickly see, and jump to, locations within the current file and in other files open in the editor. "Navigation Bar" (page 52) describes the contents of the navigation bar and how to use it to navigate source code files.

■ **Content pane.** The content pane displays the contents of the file.

> **Note:** Sometimes an editor pane may display the message "No Editor." This means that there is no file selected in the detail view or the Groups & Files list.

Navigation Bar

The **navigation bar** contains a number of controls that you can use to move between files you've viewed, jump to symbols, and open related files. Figure 4-2 shows the navigation bar.

Figure 4-2 Text editor navigation bar
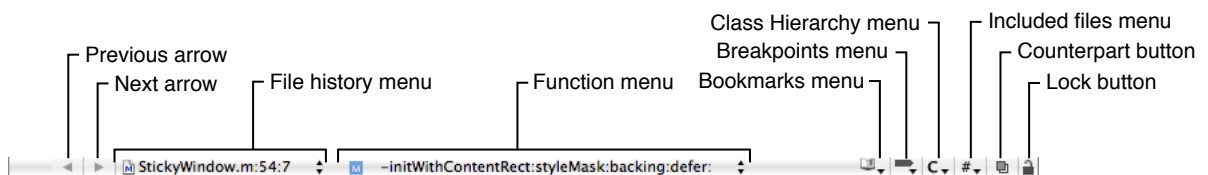
Here is what the navigation bar contains:

■ **Previous/Next arrows.** Move between open files in the editor.

■ **File History menu.** Lists the files recently viewed in the current text editor. Choosing a file from this menu displays that file in the editor, without having to repeatedly click the Next or Previous arrows.

■ **Function menu.** Shows the function and method definitions in the current file. When you choose a definition from this menu, the editor scrolls to the location of that definition. For information on how to configure the Function menu, see "The Function Menu" (page 53).

■ **Bookmarks menu.** Contains any bookmarked locations in the current file. When you choose a bookmark from this menu, the editor scrolls to the location of the bookmark. See "Defining Bookmarks" (page 36) to learn more about bookmarks in your project.

■ **Breakpoints menu.** Lists breakpoints in the current file. Choosing a breakpoint from this menu scrolls the editor to the location at which the breakpoint is set. See Managing Program Execution to learn more about breakpoints.

■ **Class Hierarchy menu.** Allows you to navigate the class hierarchy of an Objective-C class.

■ **Counterpart button.** Opens the counterpart of the current file or jumps to the symbolic counterpart of the currently selected symbol. See "Jumping to the Counterpart of a File or Symbol" (page 55) for more information on the Counterpart button.

- **Included Files menu.** Lists the files included by the current file, as well as the files that include the current file. Choosing a file from this menu opens that file in the editor. This menu is described more in "Opening Header Files and Other Related Files" (page 39).

- **Lock button.** Indicates whether the current file is editable and allows you to change the locked status of the file. (When clicked, Xcode attempts to change the file's permissions accordingly.)

The File History Menu

The **File History menu** lists the files that you have viewed in the current editor, with the current file at the top of the menu. To go to any of these files, simply choose it from the list. The menu has a couple of commands: Clear File History and History Capacity.

You can clear the File History menu by choosing Clear File History. This removes all but the current file in the editor from the list. By default, Xcode places no limit on the number of files that it places in this menu. You can limit the size of the File History menu with the History Capacity command.

The Function Menu

The **Function menu** lets you jump to many points in the current file, including any identifier it declares or defines. You can also add items that aren't definitions or declarations. In this menu, you can see:

- Declarations and definitions for classes, functions, and methods

- Type declarations

- `#define` directives

- `#pragma` marks

- Comments containing:

  - `MARK:`

  - `TODO:`

  - `FIXME:`

  - `!!!:`

  - `???:`

To scroll to the location of any of these identifiers, choose it from the menu. Figure 4-3 shows the Function menu.

Figure 4-3   The Function menu
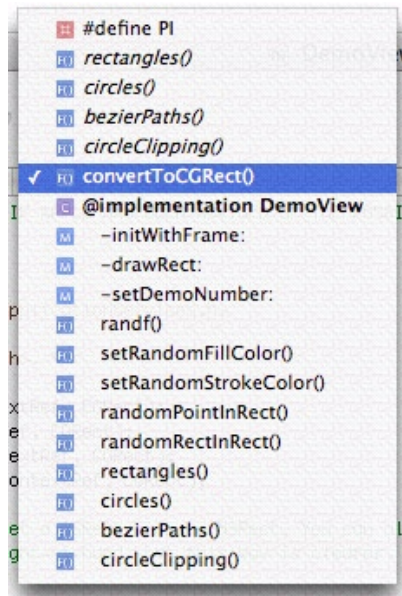
The contents of the Function menu are sorted in the order in which they appear in the file. Hold down the Option key while using the Function menu to toggle the sort order of the items between alphabetical order and the order in which they appear in the source file.

You can also change the default behavior for the Function menu in Code Sense preferences. To choose which items appear in the Function menu and the order they appear in, use:

■   The "Show declarations" option to specify whether the menu shows declarations as well as definitions

■   The "Sort list alphabetically" option to specify whether the items are sorted alphabetically or in the order they appear in the file

To add a marker to a source file and make that marker appear in the Function menu, use the `#pragma mark` statement in your source code. For example, the following statement adds "PRINTING FUNCTIONS" to the Function menu:

```
#pragma mark PRINTING FUNCTIONS
```

To add a separator to the Function menu use:

```
#pragma mark -
```

> **Supported languages:** The Function menu works with C, Objective-C, C++, Java, Perl, Python, and Ruby.

Jumping to the Counterpart of a File or Symbol

Clicking the Counterpart button opens the related header or source file for the file currently open in the text editor. For example, if the file currently open in the editor is `MyFile.c`, clicking this button opens `MyFile.h`, and vice versa. When your project contains files with the same name, Xcode gives preference to files located in the same folder as their counterparts. You can also open the current file's related header or implementation file by choosing View > Switch to Header/Source File.

Option-clicking the Counterpart button displays the counterpart of the currently selected symbol—class, method, function, and so on—opening the corresponding file and scrolling to the appropriate section within it if necessary. If the selected symbol is a class, method, or function declaration, the editor scrolls to the definition for that item. If a class, function, or method definition is currently selected, the editor scrolls to the symbol's declaration.

By default, Xcode opens the file or symbol counterpart in the current editor; however, you can have Xcode open counterparts in a separate editor window. This makes it easy to view both a header and its implementation file, or a symbol declaration and its definition, at once. To have Xcode open counterparts in a separate window, go to General preferences, and deselect the "Open counterparts in same editor" option.

Using Text Editor Windows

If you prefer, you can use a dedicated window for editing source files. Regardless of your preference for whether Xcode automatically opens the attached text editor in Xcode windows, you can always open a file in a separate editor window by doing either of the following:

■   Double-click the file in the Groups & Files list or the detail view in the project window.

■   Choose "Open in Separate Editor" from the file shortcut menu.

Figure 4-4 shows a text editor window.

Figure 4-4   The text editor in a standalone window
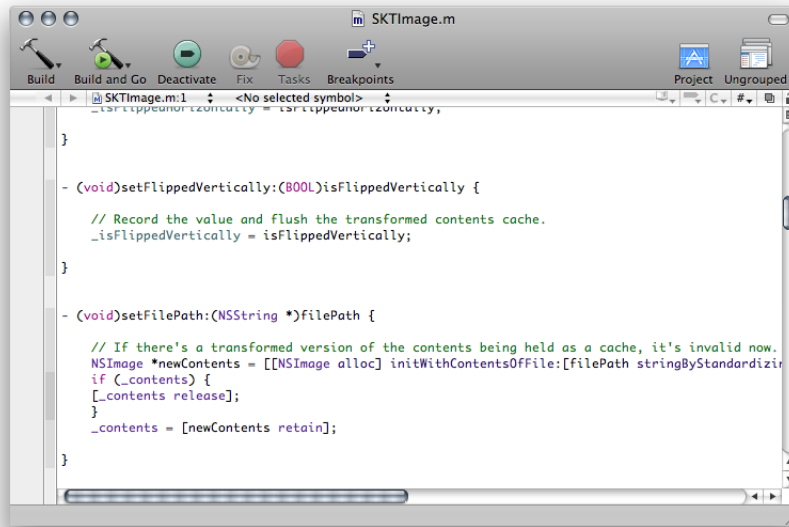
In addition to the basic editor interface, the standalone editor window also contains a toolbar and a status bar. The status bar is similar to the status bar of other Xcode windows, described in "The Project Window Status Bar" (page 20).

Like the toolbar in other Xcode windows, the toolbar in the text editor window provides easy access to common tasks. In addition to the buttons for building, running, and debugging the current target, it also contains the following buttons:

- **Breakpoints menu.** Adds breakpoints to the current file. The menu has preconfigured breakpoint actions. When you choose one of these actions Xcode adds a breakpoint at the current location of the insertion point and configures it with the specified action. See Managing Program Execution for more information.

- **Activate/Deactive button.** Toggles breakpoints on or off.

- **Project button.** Jumps to the file in the project window. Clicking this button brings the project window to the front.

- **Grouped/Ungrouped button.** Controls whether opening a file, using any of the methods described earlier in this section, opens an editor window for that file or opens the file in the current window. Clicking the button toggles the state. If the label is Grouped, indicated by the icon of a single window, double-clicking a file opens it in the current editor. If the label is Ungrouped, indicated by an icon of multiple layered windows, each file opens in a separate editor window.

To preserve the state of any open text editor windows when you close a project, select the "Save window state" option in General preferences.

Using Text Editor Panes

You can also edit your source files from within other Xcode windows, such as the project window and the debugger window, as shown in Figure 4-5. To open a file in a window's text editor pane, first make sure that the editor is visible in the window. If the editor is not already visible, you can open it by clicking Editor in the toolbar. This reveals the text editor pane of the project window. If the editor pane is at its maximum size, clicking the Editor button returns the pane to its previous size. To adjust the size of the editor pane to a different size, drag the separator to the size that you prefer. Another way to view a file in the editor pane is to select the file and choose View > Zoom Editor In.

Selecting a file, an error or warning, a bookmark, a find result or a project symbol opens the associated file in the editor pane as long as it is visible. You can also have Xcode automatically show the editor pane when you select one of these items in the detail view. Select the "Automatically open/close attached editor" option in General preferences.

Figure 4-5   The text editor pane in a project window
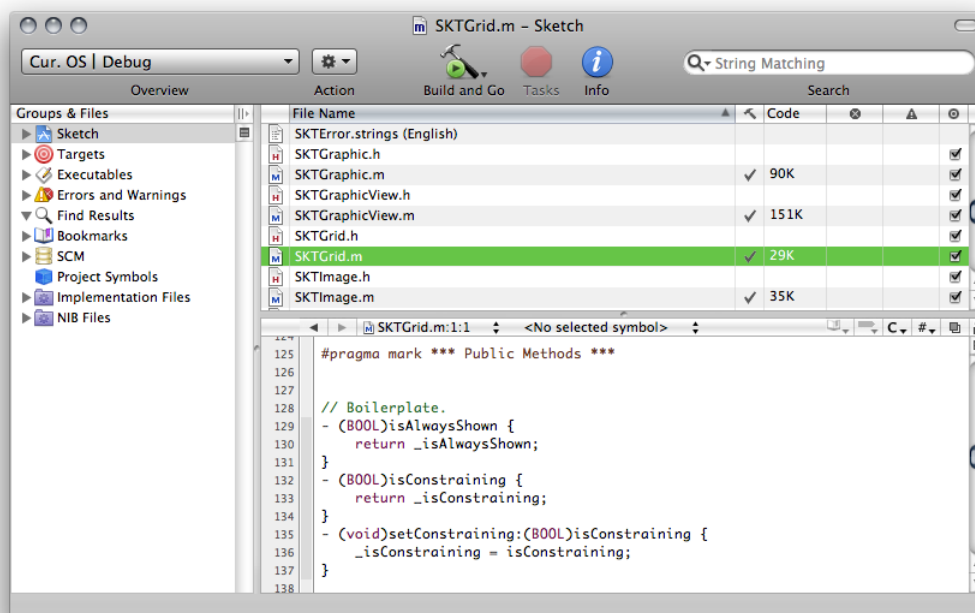
Splitting Text Editors

Xcode allows you to simultaneously view multiple files or multiple sections of the same file without increasing the number of open windows. It does this by splitting a text editor. Figure 4-6 shows an editor that has been split to display two parts of the same file.

Figure 4-6   Splitting a text editor
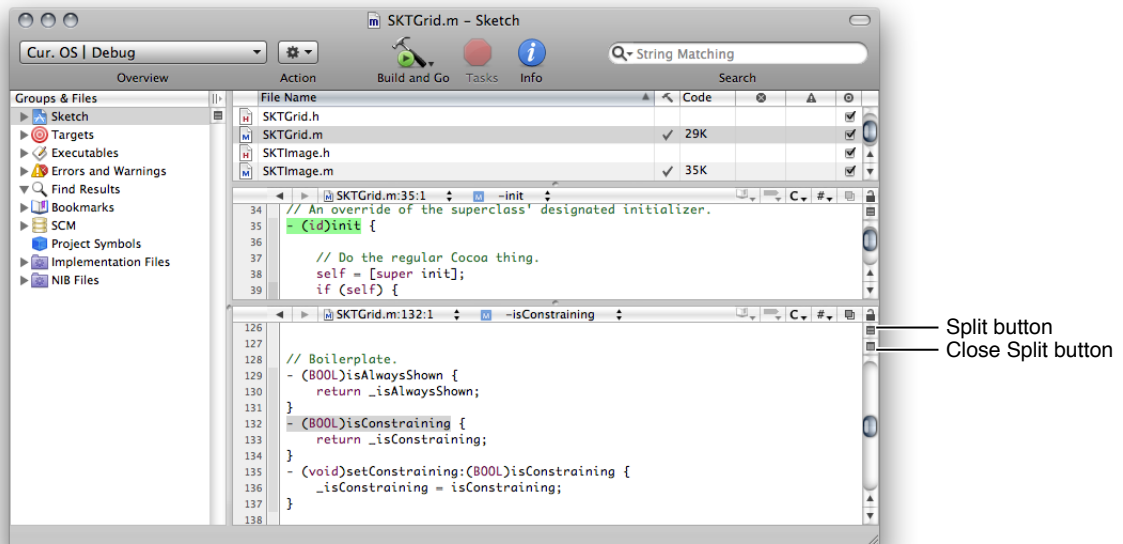
> **Note:** You can split a text editor whether that editor appears in an editor window or an editor pane.

To split a text editor, make sure that the editor has focus and do one of the following:

- To split the editor vertically click the Split button. The Split button—identical to the Split button in the Groups & Files list, described in "Splitting the Groups & Files View" (page 15)—appears above the vertical scrollbar of the editor window.

- To split a code editor horizontally Option-click the Split button.

To close a split, click the Close Split button. You can resize the panes of a split editor by dragging the resize control between them.

Laying Out Code

Xcode provides a number of layout options to help you keep your code well formed and readable. Syntax-aware indenting helps you keep your code neat by automatically indenting code as you type. This section describes options for indenting code and matching parentheses.

Indenting Code

The text editor supports syntax-aware indenting (automatic code indentation) to make it simple to author neat and readable code. You can also choose to indent code manually.

This section shows how to configure syntax-aware indenting, how to manually format text in the text editor, and how to control tab layout and automatic indentation.

Syntax-Aware Indenting

The **syntax-aware indenting** feature of the Xcode text editor gives you a number of ways to control how it automatically lays out your code. When you use syntax-aware indenting, the editor automatically indents your code as you type; pressing Return or Tab moves the insertion point to the appropriate level by examining the syntax of the surrounding lines. You can control which characters cause the editor to indent a line, what happens when you press the Tab key, and how the editor indents braces and comments.

To turn on syntax-aware indenting, select the "Syntax-aware indenting" option in Indentation preferences. See "Indentation Preferences" (page 80) for details.

Choosing What the Tab Key Does

When you use syntax-aware indenting, you usually press the Tab key to tell the editor to indent the text on the current line. But when you're at the end of the line, you may want to insert a tab character before, say, you insert a comment. To choose the circumstances for which pressing the Tab key reindents a line, use the "Tab indents" menu in Indentation preferences..

You can choose among always indenting, never indenting, or indenting only at the beginning of a line or after a space. See "Indentation Preferences" (page 80) for more information.

To insert a tab character regardless of how "Tab indents" is set, press Option-Tab. Similarly, to perform syntax-aware indenting regardless of this option's setting, press Control-I.

Choosing How to Indent Braces

You can have the editor automatically indent braces to help you easily see the level of nesting in your code and to keep your code readable. In addition, to help you keep braces balanced, you can have the editor automatically insert a closing brace when you type an opening brace.

To set how much an opening brace is indented when it appears on a line by itself, use the "Indent solo "{" by" text field in Indentation preferences.

When the value of the field is greater than 0, Xcode automatically indents opening braces to the level of the previous line plus the specified number of characters.

To specify whether to insert a closing brace automatically when you type an opening brace, select the "Automatically insert closing "}"" option.

Choosing Which Characters Reindent a Line

To set which characters cause the text editor to automatically indent a line whenever they're typed, use the options under "Automatically indented characters" in Indentation preferences. See "Indentation Preferences" (page 80) for details.

Choosing How to Indent C++-Style Comments

You can choose how to indent C++–style (//) comments when they appear on lines by themselves.

> **Note:** You cannot automatically indent C++–style comments that appear at the end of code lines.

To specify whether to indent // comments and whether to align consecutive // comments, use the options for these comments in Indentation preferences.

Manual Indenting

If you choose not to use syntax-aware indenting, you must manually indent code you want indented. When syntax-aware indenting is turned off, pressing Tab inserts a tab character and pressing Return inserts a carriage return and moves the cursor to the same level as the previous line. You can also indent a block of text to the left or right by selecting the text and choosing Edit > Format > Shift Left or Edit > Format > Shift Right.

When syntax-aware indenting is turned off, the text editor may still indent newly added lines to the level of the previous line when you press Return. You can turn this indenting off in Key Binding preferences. Click Text Key Bindings and add the Return key to the keyboard shortcuts list of the Insert Newline action. See "Keyboard Shortcuts" (page 99) for details.

Tab and Indent Layout Options

Whether you indent a line manually or rely on syntax-aware indenting, you can control the width of tabs and indents, and you can specify whether the text editor inserts tab characters or spaces. You can specify default values for all files you open in the text editor and customize these settings for individual files.

Changing the Indent and Tab Width

You can set how many spaces to indent when the editor automatically indents or when you press the Tab key. To set the default indent or tab width for every file you open, use the "Tab width" and "Indent width" text fields in Indentation preferences.

To override the default indent or tab width for one or more specific files, select the files in the Groups & Files list and open the File Info window. In the General pane, change the Indent Width or Tab Width setting.

> **Note:** If you change a file's default indent or tab width, those settings are in effect for everyone who views that file.

Using Spaces Instead of Tabs

To ensure that your code looks the same to other developers regardless of their tab layout settings, you can have the editor insert a series of spaces instead of a tab character whenever it indents code or when you press Tab.

To specify that the text editor use tabs instead of spaces, select the "Tab key inserts tab, not spaces" option in Indentation preferences..

> **Important:** When "Tab key inserts tab, not spaces" is selected, changing the width of tabs won't affect code you've already written.
>
> These options are saved in your own Xcode preferences but not in the file itself. When other people edit the file, their preferences for that file take effect.

You can also specify this setting on a per-file basis. To choose whether the editor uses tabs or spaces when editing a certain file, select the file in the Groups & Files list, open the File Info window, and in the General pane select "Editor uses tabs."

Matching Parentheses, Braces, and Brackets

Xcode provides a number of ways to help you match pairs of delimiters (parentheses, braces, and brackets):

- When you type a closing delimiter, Xcode causes its counterpart to blink.
- When syntax-aware indenting is turned on, Xcode can automatically insert a closing brace each time you type an opening brace, as described in "Choosing How to Indent Braces" (page 59).
- When you double-click any delimiter, Xcode selects the entire expression that it and its counterpart enclose. You can also choose to select the delimiters themselves.
- When you choose Edit > Format > Balance, Xcode selects the text surrounding the insertion point, up to the nearest set of enclosing delimiters.

You can further control Xcode's behavior when selecting text within a pair of enclosing braces or parentheses using the options under Editing Options in Text Editing preferences. See "Text Editing Preferences" (page 78) for details.

Wrapping Lines

To keep all your code visible in the text editor, you can have it wrap lines when they reach the right edge of the content pane. To turn on line wrapping for all files you open in the text editor, select the "Wrap lines in editor" option in Indentation preferences.

Using the "Indent wrapped lines by" option, also in Indentation preferences, you can have the text editor automatically indent wrapped lines, to visually distinguish them from other lines.

> **Note:** When "Wrap lines in editor" is not selected, the text editor does not move text to the next line until you insert a carriage return.

To wrap lines for an individual file in the current editor, choose View > Text > Wrap Lines.

Formatting Code

Syntax formatting makes it easy to identify elements of your code by using different fonts and colors to identify particular elements, such as keywords and comments. For example, you can display comments in green and keywords in boldface.

Xcode supports syntax formatting for many programming languages; to see the languages that it supports, choose View > Syntax Coloring.

The same menu allows you to toggle syntax formatting for the selected file:

- **None.** Turns syntax formatting off for the file.
- **Default for file type.** Turns syntax formatting on for the file, using the global syntax formatting settings in the Fonts & Colors pane of Xcode preferences.

To specify whether Xcode applies syntax formatting to all files that you open, select the "Use syntax-based formatting" option in Fonts & Colors preferences.

When syntax formatting is active, Xcode uses element categories to determine the formatting to apply to particular elements in a file. An element category is a name that identifies a type of source-code element or text-editor user interface. You can see a list of the categories and change the font and color for each in Fonts & Colors preferences.

Xcode provides several syntax formatting themes, which assign colors and fonts to all the element categories. You can also create your own themes.

Using its knowledge of the syntax of a programming language, Xcode assigns keywords or textual constructs in a file, such as numbers or strings (text within quotation marks) to an element category to determine how it's formatted in the editor. In addition, you can tell Xcode to use a project's Code Sense index to provide a richer store of symbol information to assign code fragments to element categories. That is, instead of only the current file, Xcode uses information about all the source files in the project to associate code fragments to categories.

For further details about configuring syntax formatting, see "Fonts & Colors Preferences" (page 83).

Completing Code

When you are writing code, you often must type out or copy and paste long identifier names and lists of arguments. **Code completion** offers you a shortcut. As you type the beginning of an identifier or a keyword, code completion suggests likely completions, based on the text you have already typed and the surrounding context within the file.

Code completion is implemented using **Code Sense**, which maintains a rich store of information about the symbols defined in your project and its included files. Xcode supports code completion for C, C++, Objective-C, Objective-C++, Java, and AppleScript. To learn more about Code Sense, see "Code Sense" in *Xcode Project Management Guide*.

This section describes how to use code completion and how to set code completion options.

> **Important:** Code completion relies on your project's Code Sense index. Code completion does not work when indexing is turned off or incomplete. See "Source Code Indexing" in *Xcode Overview* for more information.

To turn on code completion, use the Automatically Suggest menu in Code Sense preferences.

As you type, Xcode builds a list of the symbols that match the **token** (string containing no spaces) you've entered; it further modifies the list by analyzing the token's context within the file. When Xcode finds completions, the text editor shows the most likely completion inline, as shown in Figure 4-7 (the following descriptions are based on the Sketch example project).

Figure 4-7   Code completion inline
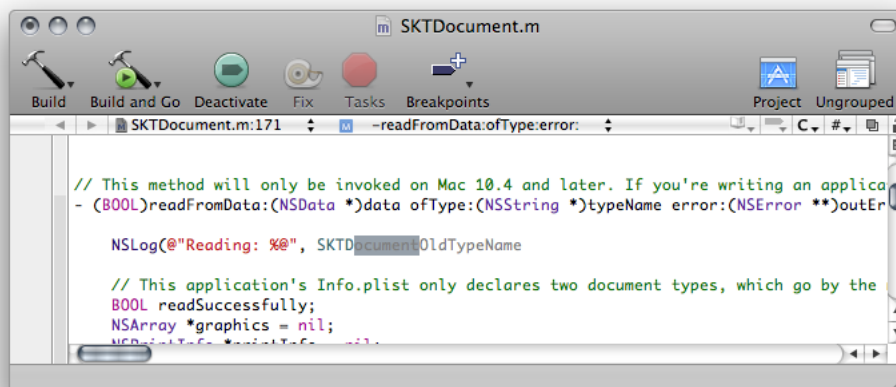
In this case, the text typed is `SKTD`. Xcode has built a list of possible completions and shown the completion that it guesses is the most appropriate, `SKTDocumentOldTypeName`. In addition the `ocument` token is highlighted because, in the list that Xcode built, all possible completions start with `SKTDocument`. Pressing Tab accepts the highlighted portion of the completion.

When the accepted completion is an indexed symbol, Xcode removes the original suggestion. In this example, `SKTDocument` is an indexed symbol; therefore, after pressing Tab, only the `SKTDocument` token remains.

With the cursor to the right of `SKTDocument`, after typing `P` Xcode suggests `SKTDocumentPrintInfoKey` because there's only one indexed symbol that begins with the token `SKTDocumentP` that would be appropriate in the token's context. Pressing Tab accepts that completion.

After accepting a completion, you can still choose the next completion in the completion list using Edit > Next Completion.

At any time, you can open the **completion list** Xcode has built so far for the typed token by choosing Edit > Completion List or pressing Escape.

Figure 4-8 shows the completion list. The button in the bottom-right corner of the list lets you specify the sort order of the completion suggestions. You can sort completions alphabetically or by relevance.

Figure 4-8   Code completion list
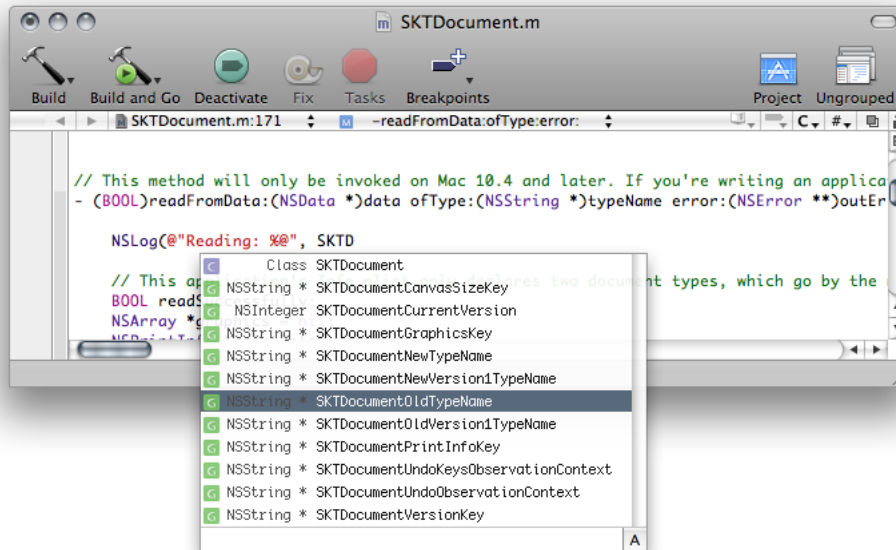
You can choose the appropriate match from this list or continue typing to narrow the list further. To enter a symbol from the completion list, select it and press Return or Tab.

For symbols that include parameters, such as methods, Xcode optionally inserts placeholders for the arguments, as shown in Figure 4-9. To move between placeholders, use Edit > Select Next Placeholder.

Figure 4-9   Code completion argument placeholders
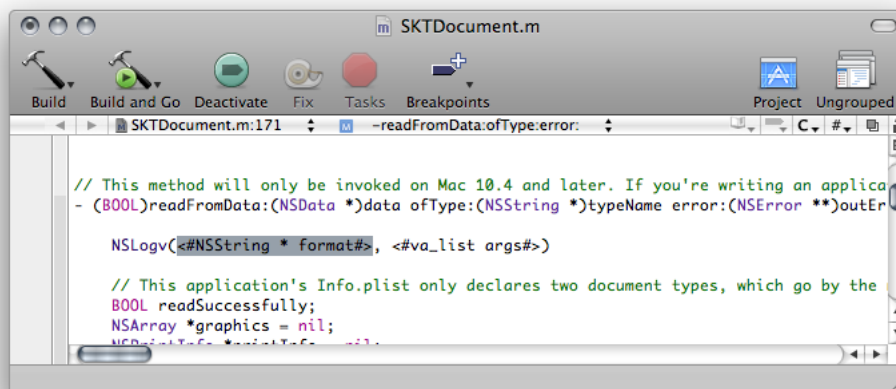
Xcode lets you specify whether and how the text editor makes code completion suggestions and how much of a symbol's information the completion list displays. You make these choices in Code Sense preferences. See "Code Sense Preferences" (page 82) for details.

You can change the keyboard shortcuts associated with code completion commands in Key Bindings preferences. Click Text Key Bindings and look for the actions that begin with "Code Sense" in the alphabetical list:

■ Code Sense Complete List

■ Code Sense Next Completion

■ Code Sense Previous Completion

■ Code Sense Select Next Placeholder

■ Code Sense Select Previous Placeholder

For more information on keyboard shortcuts, see "Keyboard Shortcuts" (page 99).

Scoping Code

The text editor provides two ways for you to focus on the part of the source file that interests you:

■ Code focus, which highlights scope levels

■ Code folding, which allows you to collapse portions of code

This section shows how to survey a source file's scope levels and how to hide areas of a file in which you're not interested.

Code Focus

Code focus highlights a source file's scope levels using a grayscale. The code at the current scope, called the **focus center**, is demarcated in the **focus box**, which uses a white background. The editor delineates subsequent scopes in boxes using progressively darker backgrounds. Figure 4-10 shows the code focus interface.

Figure 4-10 Code focus in action

Focus ribbon

Folded text

Focus box
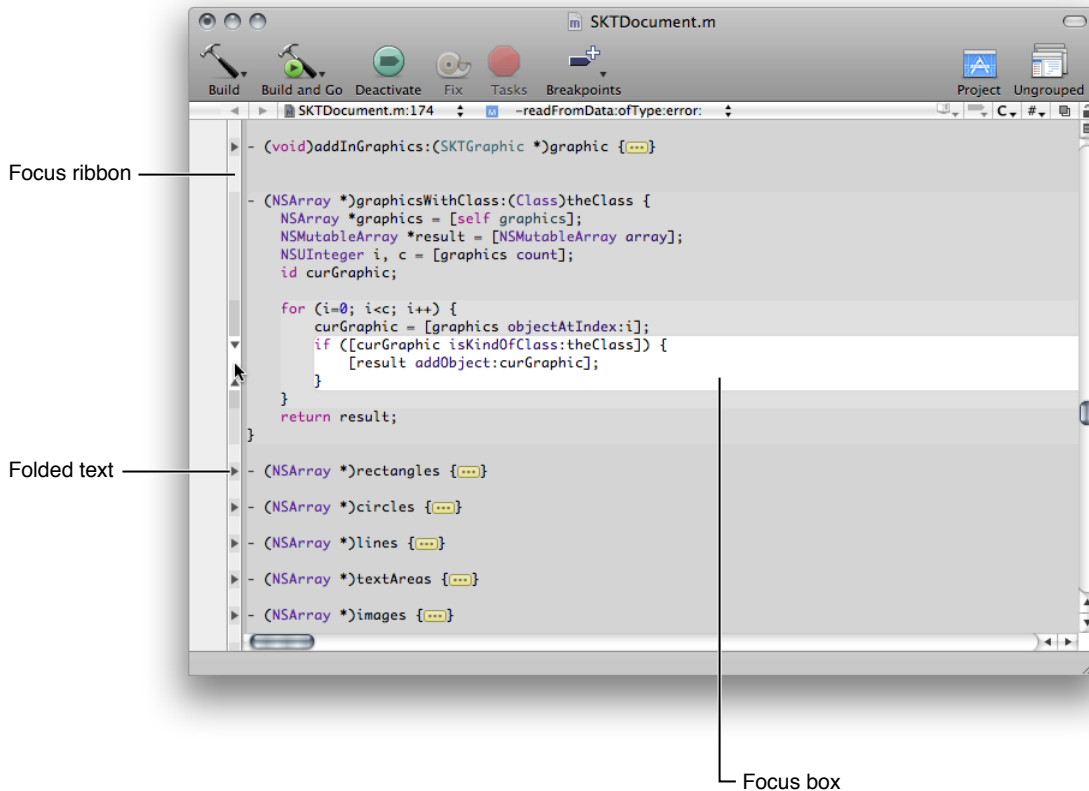
Code focus is toggled on or off in Text Editing preferences.

To survey a source file's scope levels, use the **focus ribbon**, which is located to the right of the editor gutter. It also uses levels of gray to identify the scope level of the corresponding code lines. The focus box changes as you move the pointer through the focus ribbon to show the focus center that corresponds to the pointer's position within the file.

> **Map of the code:** If you have a mouse with a scroll wheel or scroll ball, you can "fly over" your source file's scope levels by turning it up or down while the pointer remains in the same spot on the screen.

In addition to following the pointer in the focus ribbon, you can have the focus box follow the cursor's position in the content view by choosing View > Code Folding > Focus Follows Selection.

Code Folding

When you're trying to focus your attention on a specific aspect of your code, such as methods that deal with a particular instance variable, other aspects (for example, other methods or comments about your code) can get in the way. **Code folding** helps you zero in on the code you want to see by letting you hide the code you don't want to see.

In Figure 4-10 (page 66) the editor reveals only one of the methods of the SKTDocument.m file. The other methods are folded (hidden) away from view.

These are the code folding actions you can perform:

■ **Fold the focus center.** To fold the focus center, click the area in the focus ribbon that corresponds to the focus box.

■ **Unfold code.** To unfold code, click either the corresponding triangle in the focus ribbon or the icon containing the ellipsis (…) in the content pane.

You can perform other code folding actions using the Code Folding submenu of the View menu.

> **Code folding supported languages:** C, Objective-C, C++, and XML.

Editing Symbol Names

As you modify existing code or use code templates, you may want to change the name of function or method parameters to produce self-documenting code, for instance. You may also want to highlight all the places a symbol appears to see how extensively it's used within a code block. The Edit All in Scope command in the Edit menu facilitates these tasks. This command lets you select a symbol in a code block and highlight or change all occurrences of the symbol within the code block simultaneously.

For example, take this code listing:

```
- (void)setSearchKey:(NSString *) value {
    if (_searchKey != value) {
        [_searchKey release];
        if (value == nil) value = @"";
        _searchKey = [value copy];
        [self createSearchPredicate];
    }
}
```

Instead of `value` you may want to use `searchKey` to identify the method's argument. To do that, you select an occurrence of `value` anywhere within the method and choose Edit > Edit All in Scope.

You can then change `value` to `searchKey` once. Xcode replaces all occurrences of `value` with `searchKey` within the method while you type the new name, as shown in Figure 4-11.

Figure 4-11   Editing the name of a symbol

To exit editing in a scope, click anywhere in the file, outside highlighted text.

To cycle among the symbol occurrences within the code block, choose Edit > Select Next Placeholder (hold the Shift key to select the previous occurrence).

To learn how to customize the behavior of the Edit All in Scope command, see "Code Sense Preferences" (page 82).

> **Note:** The Edit All in Scope command is available only with these languages: C, C++, Objective-C, and Objective-C++.

Repeating Code

Using code completion to automatically complete symbol names saves you a lot of typing. In the course of writing source code, however, you still spend a lot of time typing the same basic code constructs—such as `alloc` and `init` methods in Objective-C programs—over and over again. To help you with this, Xcode includes a set of text macros. **Text macros** let you insert common constructs and blocks of code with a menu item or keystroke, instead of typing them in directly.

You can insert a text macro in either of these ways:

■   Choose Edit > Insert Text Macro.

Then choose a text macro from one of the language-specific submenus. Xcode provides built-in text macros for common C, C++, Objective-C, Java, and HTML constructs.

■   Type the completion prefix for the text macro ("Text Macros with Completion Prefixes" (page 70)) and use code completion (press Escape) to insert the remaining text, just as you would to complete a symbol name. Most of the text macros provided by Xcode have a completion prefix, a string that Code Sense uses to identify the text macro. When you type this string, Xcode includes the text macro in the completion list; you can select it from this list or cycle through the appropriate completions, as described in "Completing Code" (page 62).

The inserted text includes placeholders for arguments, variables, and other program-specific information. For example, choosing Insert Text Macro > C > If Block inserts the following text at the current insertion point in the active editor:

```
#if expression
 statements
#endif
```

Replace the placeholders `expression` and `statements` with your code. You can cycle through the placeholders in a text macro in the same way you can cycle through function arguments with code completion. A text macro can also define one placeholder to be replaced with the current selection, if any. When you select text in the active editor and insert a text macro, Xcode substitutes the selected text for this placeholder. For the If Block text macro described above, Xcode substitutes the selected text for the `statement` placeholder. For example, if the current selection in the text editor is `CFRelease(someString);`, inserting the If Block text macro gives you the following:

```
#if expression
CFRelease(someString);
#endif
```

If there is no selection, Xcode simply inserts the `statements` placeholder, as in the previous example.

Some text macros have several variants. For example, the text macro for inserting an HTML heading has variants for the different levels of headings. For text macros that have multiple variants, repeatedly choosing that text macro from the Insert Text Macro menus cycles through the different versions of that macro. For example, choosing Insert Text Macro > HTML > Heading a single time produces:

```
<h1> text </h1>
```

Choosing it again modifies the text to:

```
<h2> text </h2>
```

To create your own text macros, you have to create a language specification for the language to which you want to apply the macro. Then, place the language specification in:

```
~/Library/Application Support/Developer/Shared/Specifications
```

For examples of language specifications, see:

```
<Xcode>/Applications/Xcode.app/Contents/PlugIns/TextMacros.xctxtmacro
```

Text Macros with Completion Prefixes

The following tables list the built-in text macros with completion prefixes for C, Objective-C, and C++.

Table 4-1   C text macros with completion prefixes

Text macro name	Completion prefix
If Block	if
If / Else Block	ifelse
Else If Block	elseif
For Loop	for
For i Loop	fori
While Loop	while
Do While Loop	do
Switch Block	switch
Case Block	case
Else Block	else
Enum Definition	enum
Struct Definition	struct
Union Definition	union
Type Definition	typedef
Printf() Call	printf
#Pragma Mark	pm
Pragma Mark	pragma
#Import Statement	pim
#Import Statement (System)	pims
#Import Statement (Framework)	pimf
#Include Statement	pin
#Include Statement (System)	pins
#If Block	pif
#Ifdef Block	pifd

Text macro name	Completion prefix
#if / Else Block	`pife`
#Ifdef / Else Block	`pifde`
#if 0 Block	`pifz`
Copyright Comment	`copyright`
Comment Selection	`comment`
Separator Comment	`cseparator`

Table 4-2   Objective-C text macros with completion prefixes

Text macro name	Completion prefix
Try / Catch Block	`@try`
Catch Block	`@catch`
Finally Block	`@finally`
NSLog() Call	`log`
Alloc / Init Call	`a`
Array Declaration	`aa`
Mutable Array Declaration	`ma`
Array For Loop	`fora`
Array Foreach Loop	`fore`
init Definition	`init`
dealloc Definition	`dealloc`
observeValueForKeyPath: Definition	`observeValueForKeyPath`
observeValueForKeyPath: Declaration	`observeValueForKeyPath`
bind: Definition	`bind`
bind: Declaration	`bind`
@interface Definition	`@interface`
@implementation Definition	`@implementation`
@protocol Definition	`@protocol`
NSString	`nss`

Text macro name	Completion prefix
NSMutableString	`nsms`
NSArray	`nsa`
NSMutableArray	`nsma`
NSDictionary	`nsd`
NSMutableDictionary	`nsmd`

Table 4-3   C++ text macros with completion prefixes

Text macro name	Completion prefix
#Ifdef Block	`pifdcpp`
#Ifdef/Else Block	`pifdecpp`
Static Cast	`static_cast`
Dynamic Cast	`dynamic_cast`
Reinterpret Cast	`reinterpret_cast`
Try / Catch Block	`try`
Catch Block	`catch`
Cout Statement	`cout`
Cout Message	`coutm`
Namespace Definition	`namespace`
Class Definition	`class`
Extern \"C\" Statement	`extern`
Extern \"C\" Block	`externb`

Viewing Project Messages

To help keep your attention on as few windows as possible, Xcode can display some information generated as you work on a project in the text editor. The editor displays this information through message bubbles. **Message bubbles** display project messages in a concise way in the code line from which they are generated (in the case of build messages) or at which they are placed (in the case of breakpoints). You can view and dismiss build messages and modify breakpoints without taking your focus away from the file you're editing in the text editor.

Figure 4-12 shows a build message bubble.

Figure 4-12   A build message bubble
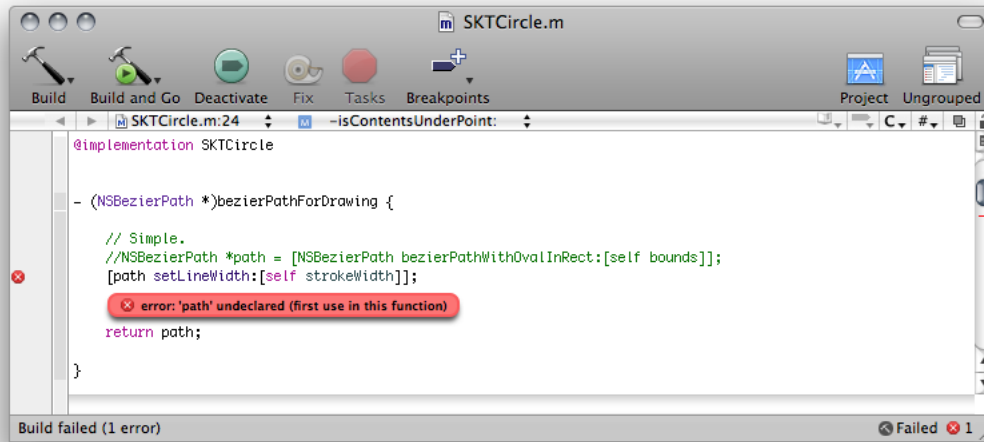
To show or hide message bubbles in the text editor, use the Message Bubbles submenu of the View menu.

The Message Bubbles submenu also allows you to display and hide message bubbles as well as specify the kind of messages in which you're interested.

Executing Shell Commands in Selection

Xcode provides a keyboard shortcut for executing any shell command that appears in a text editor window. To use this feature, select the command text and press Control-R. The results appear below the command in the text editor window, automatically scrolling if necessary to show the output.

Xcode creates a shell each time you execute a command, so there is no shared context between different executions. For example, if you execute a command that changes the directory, the next command you execute does not execute in that directory. To overcome this, you can select two commands and press Control-R.

One way to take advantage of this feature is to keep a file of commonly used commands and execute them as needed. Or you might use an empty text file as a scratch area to type and execute commands.

You can also add custom menu items to execute frequently used shell scripts. Any scripts you execute can take advantage of many special script variables and built-in scripts defined by Xcode. For more information, see "User Scripts" (page 103).

Searching Source Files

The Xcode text editor provides many ways to move between files and to find and navigate to information in a file. The navigation bar of the editor provides a number of menus for navigating within and between files. See "Navigation Bar" (page 52) for details.

This section shows how to search within a file. It also shows some of the shortcuts you can use in the text editor to find text and symbol definitions.

Some tricks for finding information from an editor are described in other chapters. "Opening Files by Filename or Symbol Name" (page 40) describes shortcuts you can use to open a file whose name or path appears in an editor. "Searching Documentation" (page 90) describes shortcuts you can use to jump to the documentation for a symbol whose name appears in the editor or search the installed documentation for a word or phrase.

Searching in a File

When editing files in the text editor, it is fairly common to want to make the same change in several places in the current file. For example, when you rename a function, you also have to find all of the places where you call that function and change those calls to use the new name. Xcode provides a single-file find that allows you to search for and replace text within a single file. You can perform a simple textual search or search using regular expressions.
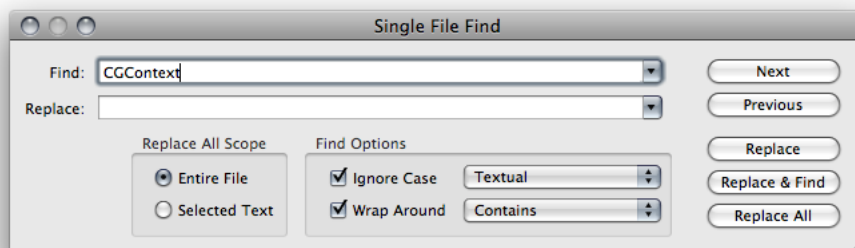
> **Tip:** Xcode refactoring provides a more efficient way of renaming a symbol throughout multiple files. See *Xcode Refactoring Guide* for details.

Specifying Search Terms

To search for text in a file that you have open in the editor, choose Edit > Find > Single File Find.

Xcode displays the Single File Find window, shown in Figure 4-13.

Figure 4-13   Single File Find window

You can search using a text string or a regular expression; choose the appropriate search type from the pop-up menu next to the Ignore Case option in the Find Options group. Choosing Textual searches for text matching the string in the Find field. Choosing Regular Expression searches for text matching the regular expression in the Find field; Xcode uses the ICU library for regular expression matching.

Type the text string or regular expression pattern to use for the search in the Find field. To minimize your typing, Xcode keeps track of search strings; to reuse a previous search string, click the arrow in the Find field and choose the string from the menu.

The other options in the Find Options group give you additional control over how the search is performed; these options are:

■ **Ignore Case.** Select this option to ignore whether letters are uppercase or lowercase.

■ **Wrap around.** Select this option to search the whole file; otherwise, Xcode searches from the current location of the insertion point to the end of the file.

■ The pop-up menu next to the Wrap Around option specifies how Xcode determines a match to the search term in the Find field:

   ❏ **Contains.** Searches for words that contain matching text in a substring.

   ❏ **Starts with.** Searches for words that begin with text matching the search term.

   ❏ **Whole words.** Searches for words that contain only text matching the contents of the Find field.

   ❏ **Ends with.** Searches for words that end with matching text.

Use the Next and Previous buttons to continue searching for the same text in a file. You can also use menu commands. Choose Edit > Find > Find Next to move to the next match; to move to the previous match, hold down the Shift key while choosing that sequence.

Pressing Return finds the next (or first) match and dismisses the Single File Find window.

Replacing Text

You can use the Single File Find window to replace some or all occurrences of text matching the string or regular expression specified in the Find field.

To search for and replace text in a file:

1. Open the Single File Find window and specify the search criteria, as described in "Specifying Search Terms" (page 74).

2. Type the replacement text in the Replace field. As with search strings, Xcode keeps track of substitution strings; to reuse a previous substitution string, choose it from the menu in the Replace field.

Use the replace buttons in the bottom-right portion of the Single File Find window to perform the text substitution. The scope of the replacement varies, depending on the button you choose. Here are the buttons available to you:

■ **Replace.** Substitutes the replacement text for the current selection.

■ **Replace & Find.** Substitutes the replacement text for the current selection and then finds and selects the next occurrence of text matching the contents of the Find field.

■ **Replace All.** Searches the entire file or selection and replaces all occurrences of text matching the contents of the Find field with the replacement text.

If you use Replace All, the Replace All Scope options control the scope of the search and replace operation. To search for and replace instances of the given search text throughout the entire file, select the Entire File option. To perform the search and replace operation within only the current selection, choose Selected Text.

Each of these buttons also has a menu item equivalent in the Find submenu: Replace, Replace All, and Replace and Find Next. Holding down the Shift key changes Replace and Find Next to Replace and Find Previous.

Shortcuts for Finding Text and Symbol Definitions

Xcode provides a number of shortcuts for searching that use text or other content that appears in the text editor. You can use these shortcuts to perform single-file and projectwide searches without going through the Single File Find or Project Find windows. When you use these shortcuts, Xcode performs the search using the same options specified the last time you used the Single File Find or Project Find windows. These windows are described in further detail in "Searching in a File" (page 74) and "Searching in a Project" in *Xcode Project Management Guide*.

To search the current project for text that appears in the text editor, select the text to search for, and choose Edit > Find > Find Selected Text.

To perform a projectwide search using the current selection in the text editor, use the shortcuts listed in Table 4-4.

Table 4-4   Shortcuts for performing a projectwide search using the current selection in the editor

Search project for	Choose
Selected text	Edit > Find > Find Selected Text in Project
Selected symbol definition	Editor shortcut menu > Find > Find Selected Definition in Project

You can also jump directly to the definition for a symbol identifier by doing either of the following:

■ Command—double-click the symbol name.

■ Select the symbol name and choose Edit > Find > Jump to Definition.

Each of the searches described in Table 4-4 (page 76) uses the last set of search options used when searching your project. If you want to perform a projectwide search using the current selection in the text editor, but do not want to use the last set of search options, you can open a Project Find window with the current selection by doing either of the following:

■ To use the current selection as the search term, choose Edit > Find > Use Selection for Find.

Xcode opens a Project Find window and places the contents of the current selection in the Find field.

■ To use the current selection as a substitution string, choose Edit > Find > Use Selection for Replace.

Xcode opens the Project Find window and places the contents of the current selection in the Replace field.

Customizing the Editor

Xcode gives you a great deal of flexibility to customize the appearance of the editor. You can change the fonts and colors used to display text in the editor to suit your own preferences. You can also control the amount of information that Xcode displays about file locations and contents. This section describes how to change the default font and text editing colors for the text editor, and how to use the gutter, page guide, and file history menu to locate information in a file.

Displaying a Page Guide

To help keep code lines no longer than a specified length, you can have Xcode display a guide line in the editor at that column position in the text view. You activate the guide line in Text Editing preferences. Select "Show page guide" option. Enter the location, in number of characters, at which you want the guide line displayed in the "Display at column" field. Xcode displays a gray line in the right margin of the text editor, at the specified column.

Xcode does not wrap your code lines when they reach the guide line. The line only serves as a visual cue.

Displaying the Gutter

The **gutter** that appears on the left side of the content pane in the editor helps you quickly locate items in a file. The gutter can display:

- **Line numbers for the current file.** Line numbers make it easy to find a location in a file. To show line numbers, select the "Show line numbers" option in Text Editing preferences.

- **Errors and warnings.** To help you locate and fix problems in your code, Xcode displays error and warning icons next to the line at which an error or warning occurred. Clicking the icon or hovering the pointer over the icon displays the error or warning message.

- **Breakpoints.** You can use the gutter to set, remove, and otherwise control the breakpoints in a file. Xcode indicates the location of a breakpoint by displaying an arrow next to the line at which the breakpoint is set. For more information on using breakpoints, see "Using Breakpoints" in *Xcode Debugging Guide*.

To show or hide the gutter in the text editor, use the "Show gutter" option in Text Editing preferences.

Viewing Column and Line Positions

The File History menu in the navigation bar not only lets you move between currently open files, it also shows your current location in the file. The File History menu shows the name of the current file and the line number of the line containing the insertion point. You can also have the File History menu display the column position of the insertion point. Figure 4-14 shows the location of the current insertion point in the File History menu.

Figure 4-14   Line numbers and column positions in the File History menu
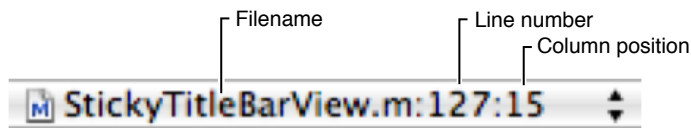
To set whether the text editor displays the column position of the insertion point, use the "Show column position" option in Text Editing preferences.

Text Editing Preferences

The Text Editing pane of Xcode preferences contains options that control the appearance and behavior of the text editor. Figure 4-15 shows Text Editing preferences.

Figure 4-15   Text Editing preferences pane
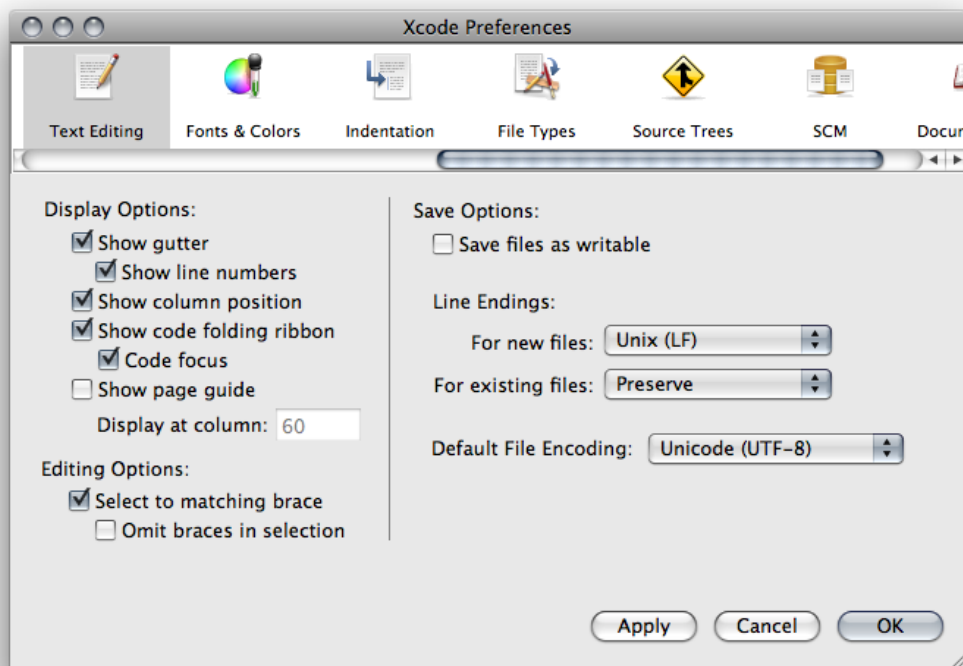
Here is what the pane contains:

■ **Display Options.** These options control the appearance of the text editor, whether it appears as a separate window or as a pane attached to another Xcode window. See "Customizing the Editor" (page 77) to learn more about changing the appearance of Xcode's editor. The options are:

❑ **Show gutter.** Specifies when Xcode displays the gutter in the editor. The gutter shows information about the current file such as the location of breakpoints, line numbers, and the location of errors or warnings. If this option is selected, Xcode always shows the gutter in all open editors; otherwise, it shows the gutter only when debugging. See "Displaying the Gutter" (page 77) for more information.

❑ **Show line numbers.** Specifies whether Xcode shows a file's line numbers in the editor gutter. If this option is selected, Xcode shows line numbers for a file whenever the editor gutter is visible. See "Displaying the Gutter" (page 77) for more information

❑ **Show column position.** Specifies whether Xcode shows the current position of the cursor in the Function menu of the editor. If this option is selected, Xcode shows the character position of the insertion point along the current line. See "Searching Source Files" (page 74) for more information.

❑ **Show code folding ribbon.** Specifies whether the text editor shows the folding ribbon used to view a source file's scope levels and to perform code folding operations. See "Scoping Code" (page 65) for more information.

Code focus. Specifies whether code focus is active. See "Code Focus" (page 65) for details.

❑ **Show page guide.** Specifies whether Xcode displays the page guide in the editor. If this option is selected, Xcode displays a gray guide line in the editor to show you the right margin of the editor; to the right of this margin, Xcode colors the background of the editor light gray. This is just a guide, and does not actually affect the margin width in the editor. See "Displaying a Page Guide" (page 77) for more information.

Display at column. Controls the column position at which Xcode displays the page guide. This position is specified in number of characters. To change the position at which the page guide is shown, enter a new number in the field. See "Displaying a Page Guide" (page 77) for more information.

■ **Editing Options.** These options control Xcode's selection behavior for source code. The options are:

❑ **Select to matching brace.** Specifies whether Xcode automatically selects text contained in braces when you double-click the brace. If this option is selected, double-clicking a brace, bracket, or parenthesis in a source code file automatically selects the text up to, and including, the matching brace. See "Matching Parentheses, Braces, and Brackets" (page 61).

❑ **Omit braces in selection.** Specifies whether Xcode includes the braces themselves in text selected by double-clicking a brace, bracket, or parenthesis. If this option is selected, double-clicking a brace, bracket, or parenthesis selects the text between the braces, but not the braces themselves. See "Matching Parentheses, Braces, and Brackets" (page 61) for more information.

■ **Save Options.** These options let you specify how Xcode stores files that you edit in the text editor.

❑ **Save files as writable.** Specifies the permissions that Xcode uses for files that it saves. If this option is selected, Xcode adds write permission to files that you edit and save in Xcode. Otherwise, Xcode preserves permissions for files as they are on disk. Files that you create in Xcode already have write permission.

❑ **Line Encodings.** Controls the default line endings used for files in Xcode. You can use Unix, Windows, or Mac line endings for files that you open and edit in the text editor; the type of line endings used for a file can affect which file editors and other tools can interpret the file. See "Changing Line Endings" (page 48) for more information on line endings. The menus are:

❑ **For new files.** Specifies the type of line endings used for files that Xcode creates. You can choose Unix, Mac, or Windows line endings. The default value for this setting is Unix.

❑ **For existing files.** Specifies the type of line endings used for preexisting files that you open and edit in Xcode. If you choose Unix, Mac, or Windows from this menu, Xcode saves all files that you open and edit in Xcode with line endings of this type, changing them the next time it saves the file, if necessary. If you choose Preserve from this menu, Xcode uses whatever type of line endings the file already has.

❑ **Default File Encoding.** Specifies the default file encoding that Xcode uses for new files that you create in Xcode. You can choose any of the file encoding supported by your Mac OS X system from this menu. See "Choosing File Encodings" (page 47) for more information.

Indentation Preferences

The Indentation pane of Xcode preferences controls formatting options for files in the text editor. Figure 4-16 shows the Indentation preferences pane.

Figure 4-16   Indentation preferences pane
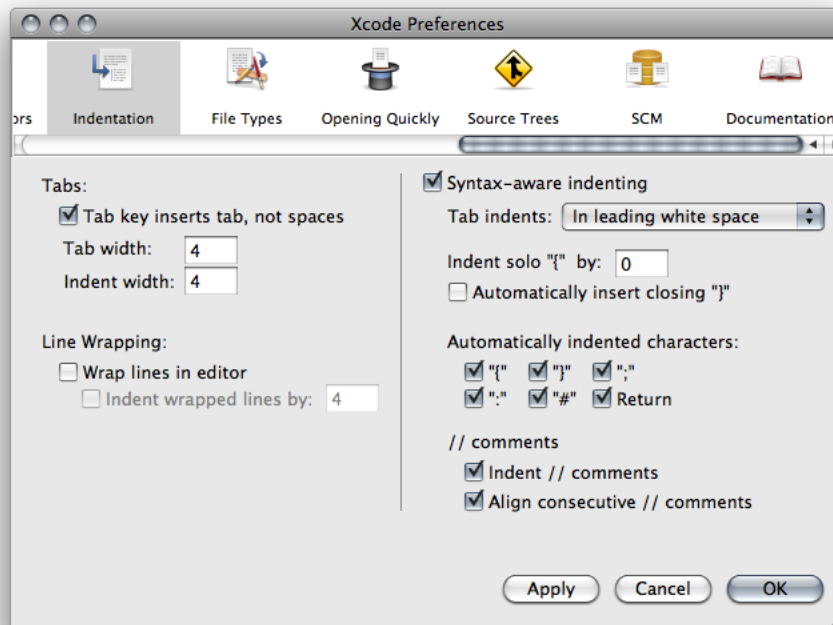
Here is what the pane contains:

■ **Tabs.** These options specify how the text editor inserts space into files while editing (see "Tab and Indent Layout Options" (page 60) for more information):

❑ **Tab key inserts tab, not spaces.** Specifies whether Xcode inserts tab characters when you press the Tab key in the editor. If this option is selected, pressing the Tab key inserts a Tab character. Otherwise, Xcode inserts space characters.

❑ **Tab width.** Specifies the default width, in number of characters, used to display tabs in the editor. To change the width of a tab, type a number in the text field. You can override this setting for individual files, as described in "Tab and Indent Layout Options" (page 60).

❑ **Indent width.** Specifies the default width, in number of characters, used to indent lines in the editor. To change the indentation width, type a number in the text field. You can override this setting for individual files, as described in "Tab and Indent Layout Options" (page 60).

■ **Line Wrapping.** These options specify how the text editor wraps lines in files displayed in the editor. These options affect how the file is displayed onscreen, not line breaks or other information stored with the file. See "Wrapping Lines" (page 61) for more information. The options are:

❑ **Wrap lines in editor.** Specifies whether the editor wraps lines. If this option is selected, the editor wraps text to the next line when it reaches the outer edge of the text editing area onscreen. Otherwise, Xcode moves text to the next line only when a carriage return or new line characters is inserted.

❑ **Indent wrapped lines by.** Specifies how the editor indents text that it wraps to the next line. If this number is greater than 0, the editor indents wrapped text by the specified number of characters, as a visual indication that the text has been wrapped (as opposed to being moved to the next line by the insertion of a carriage return or new line character). To change the amount by which lines are indented, type a new number in the field.

■ **Syntax-Aware Indenting.** This option, and the options below it, control automatic formatting for source code in the text editor. If this option is selected, Xcode assists you in writing source code by automatically inserting formatting information appropriate for the current context. See "Indenting Code" (page 58) to learn more. The options are:

❑ **Tab indents.** Specifies when pressing Tab in the editor inserts an indentation. You can choose the following:

In leading white space: Pressing Tab inserts an indentation only at the beginning of a line or following a space.

Never: Pressing Tab never causes an indentation.

Always: Pressing Tab always causes an indentation.

❑ **Indent solo "{" by.** Controls the amount by which a left brace character ({) on a line by itself is indented. If this number is greater than 0, Xcode automatically indents a left brace that appears on a line by itself (that is, a left brace that is preceded by a newline or carriage return) by the specified number of characters. The default value of this field is 0.

❑ **Automatically insert closing "}".** Controls whether Xcode automatically inserts a matching right brace when you type an opening brace. If this option is selected, typing an opening brace causes Xcode to insert a matching closing brace.

❑ **Automatically indented characters.** Controls which characters trigger Xcode to automatically cause an indentation. When any of the following options is selected, typing that character in an editor causes Xcode to indent the current line or the following line.

❑ **Indent // comments.** Controls whether Xcode automatically indents C++-style comments. If this option is selected, Xcode automatically indents comments beginning with //.

❑ **Align consecutive //comments.** Controls whether Xcode automatically indents consecutive C++-style comments to the same level.

Code Sense Preferences

The Code Sense pane of Xcode preferences contains options for controlling symbol indexing, the text editor's Function menu, and its code completion interface. Figure 4-17 shows the Code Sense pane. For more information about Code Sense, see "Code Sense" in *Xcode Project Management Guide*.

Figure 4-17   Code Sense preferences pane
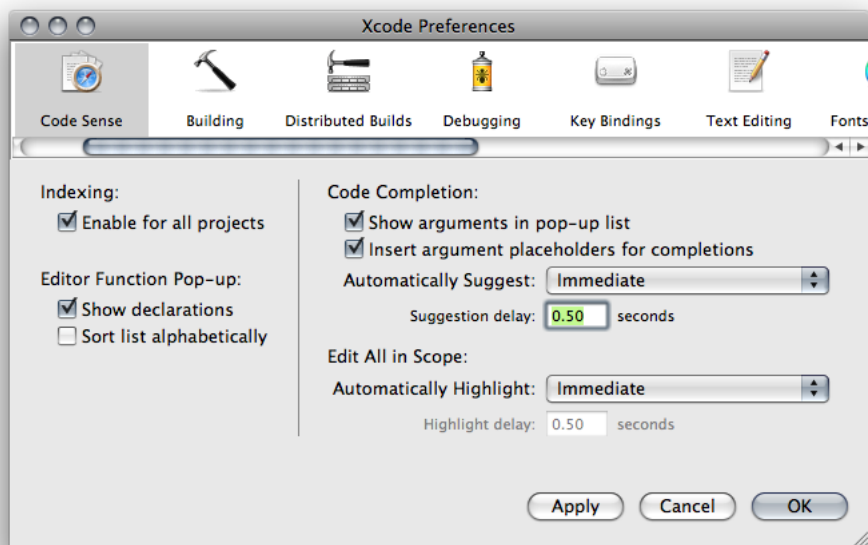
Here's what the pane contains:

- **Indexing.**
    - ❑ **Enable for all projects:** Specifies whether symbol indexing is active for the projects you open.

        When inactive, features that rely on this index (such as the Project Symbols smart group, refactoring, and code completion) do not work. For more information about symbol indexing, see "Code Sense" in *Xcode Project Management Guide*.

- **Editor Function Pop-up.** This area contains options for configuring the text editor Function menu ("The Function Menu" (page 53)).
    - ❑ **Show declarations.** Specifies whether the Function menu shows function and method declarations (in addition to definitions).
    - ❑ **Sort list alphabetically.** Specifies whether the contents of the Function menu are sorted alphabetically.

        When unselected, the Function menu shows items in the order they appear in the file.

- **Code Completion.** This area contains options for configuring code completion. For more information about code completion, see "Completing Code" (page 62). (Your code completion settings apply to all projects that you open.)

❑ **Show arguments in pop-up list.** Specifies whether Xcode displays arguments for functions and methods in the completion list. When this option is selected, Xcode displays the return type and arguments for functions and methods in the list of completion suggestions. Otherwise, Xcode shows only the symbol name.

❑ **Insert argument placeholders for completions.** Specifies whether Xcode inserts the arguments to a function or method when you accept a completion suggestion. When selected, inserting a function or method using code completion also inserts placeholders for arguments. Otherwise, Xcode inserts only the symbol name.

❑ **Automatically Suggest.** Specifies whether and when Xcode shows completion suggestions. You can choose between immediate and delayed suggestions.

❑ **Suggestion delay.** The number of seconds the text editor waits before showing its best completion suggestion when using delayed completion suggestions.

■ **Edit All in Scope.** This area contains options for configuring Edit All in Scope behavior (see "Editing Symbol Names" (page 67) for details about this feature).

❑ **Automatically Highlight:** Specifies whether and when Xcode highlights the occurrences of a symbol whose name can be edited using Edit All in Scope. You can choose between immediate and delayed suggestions.

❑ **Highlight delay.** The number of seconds the text editor waits before highlighting symbol occurrences when using delayed highlighting.

Fonts & Colors Preferences

The Fonts & Colors pane of Xcode preferences (Figure 4-18) is where you configure syntax formatting, which identifies elements of the source files you edit in the text editor (see "Formatting Code" (page 62) for details).

Figure 4-18   Fonts & Colors preferences pane
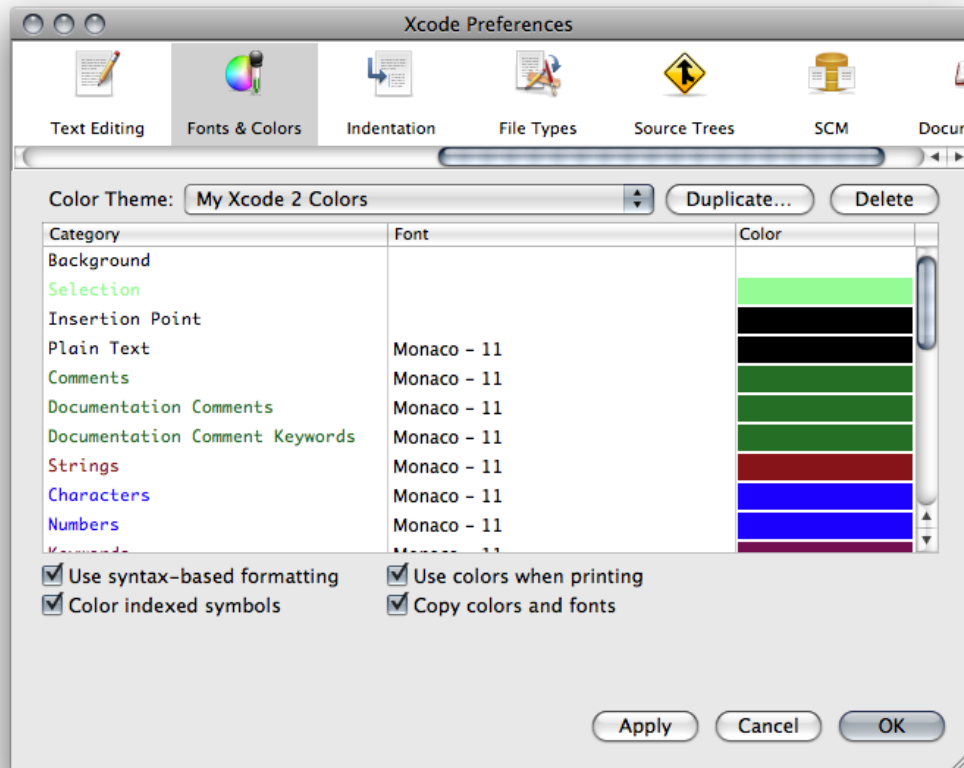
Here is what the pane contains:

■ **Color Theme:** Lists of syntax formatting themes. Each theme applies a set of fonts and colors to all the syntax formatting element categories Xcode supports.

■ **Duplicate:** Duplicates the current syntax formatting theme.

■ **Delete:** Deletes the current syntax formatting theme.

■ **Category list:** List of the syntax formatting element categories.

To change the font or color for a category, double-click the font or color you want to change to display the Fonts window or the Colors window.

You can more than one category to modify the font and color for a set of categories at the same time.

> **Note:** The Documentation Comments and Documentation Keywords categories apply to HeaderDoc and JavaDoc comments, and their @-based keywords, respectively.

■ **Use syntax-based formatting:** Activates syntax formatting.

■ **Color indexed symbols:** Specifies whether to use the project's Code Sense index to assign symbols in source files to element categories. When turned off, the text editor uses only the file's language to assign symbols and text to categories.

- **Use colors when printing:** Preserves syntax formatting when you print source files from Xcode.

- **Copy colors and fonts:** Places syntax formatting on the Clipboard when you copy text from the text editor.

> **Note:** Before you can customize the formatting of text categories, you must create a custom color theme by duplicating one of the predefined themes.

Documentation Access

Documentation is an important resource of the software development process. As you develop products with Xcode, you're likely to use documentation to learn about Apple technologies, read about system frameworks, and look up API reference. You may also need to consult documentation provided by other vendors—for example, if you use a third-party framework to develop an application.

Xcode provides two ways to access documentation:

- The Research Assistant
- The Documentation window

The **Research Assistant** is a lightweight window that provides a condensed view of the API reference and links to related documentation for the selected system or build setting without taking the focus away from the editor in which the item is located. The Research Assistant allows you to continue editing code or a build setting's value while viewing relevant documentation at the same time.

The **Documentation window** lets you browse and search the developer documentation installed on your computer. It provides access to a wider and more detailed view of the documentation than the Research Assistant.

The Apple Developer Connection (ADC) Reference Library contains all the developer documentation Apple publishes. You can view the entire library on the web at developer.apple.com.

To reduce Xcode's documentation footprint on your computer, only guides and API reference are placed in your file system. However, the Documentation window displays the titles of all document types. Xcode opens nonresident documents in your web browser (you must have a working connection to the Internet to access such documents).

The Documentation window lets you view **documentation sets** (also known as *doc sets*). Each doc set contains HTML-based content as well as indexes into that content, which Xcode uses to perform quick documentation searches. When searching for documentation in the Documentation window, you can search a single doc set—resulting in fast, focused searches—or all doc sets.

These are some of the doc sets that Apple publishes:

- Developer Tools Reference Library
- Core Reference Library
- Java Reference Library
- WebObjects Reference Library

When you install Xcode, you can choose which doc sets to place on your file system (except for the Developer Tools Library doc set, which is always installed).

This chapter discusses the documentation-viewing experience Xcode provides through the Documentation window and the Research Assistant. It describes how to tell Xcode where to get the documentation it displays, how to perform documentation searches, and how to view man pages for the UNIX developer tools installed in your file system.

Viewing Documentation

The Documentation window lets you view HTML-based documents that are part of the documentation sets installed on your computer. (Xcode opens PDF-based documents in your PDF-file viewer.) The Documentation window also provides access to documents known to a doc set but stored elsewhere.

To open the Documentation window, choose Help > Documentation.

The Documentation window operates in two modes: browse and search. In **browse mode** you can browse the documents in a doc set using its category hierarchy. In **search mode** you can find documents by API symbol name, document title, or document content.

In browse mode you select documents in the documentation browser view. In search mode you select them in the documentation detail view. In both modes, you view documentation in the content pane, which displays category information or a document page. To display the current page in another window or in your web browser, use the content pane's shortcut menu. You can use the hyperlink shortcut menu to open cross-referenced documents in another window or in your web browser, as well.

Browsing Documentation

Figure 5-1 shows the Documentation window in browse mode. It identifies the documentation-set list, the browser view, and the content pane.

Figure 5-1   Documentation window in browse mode
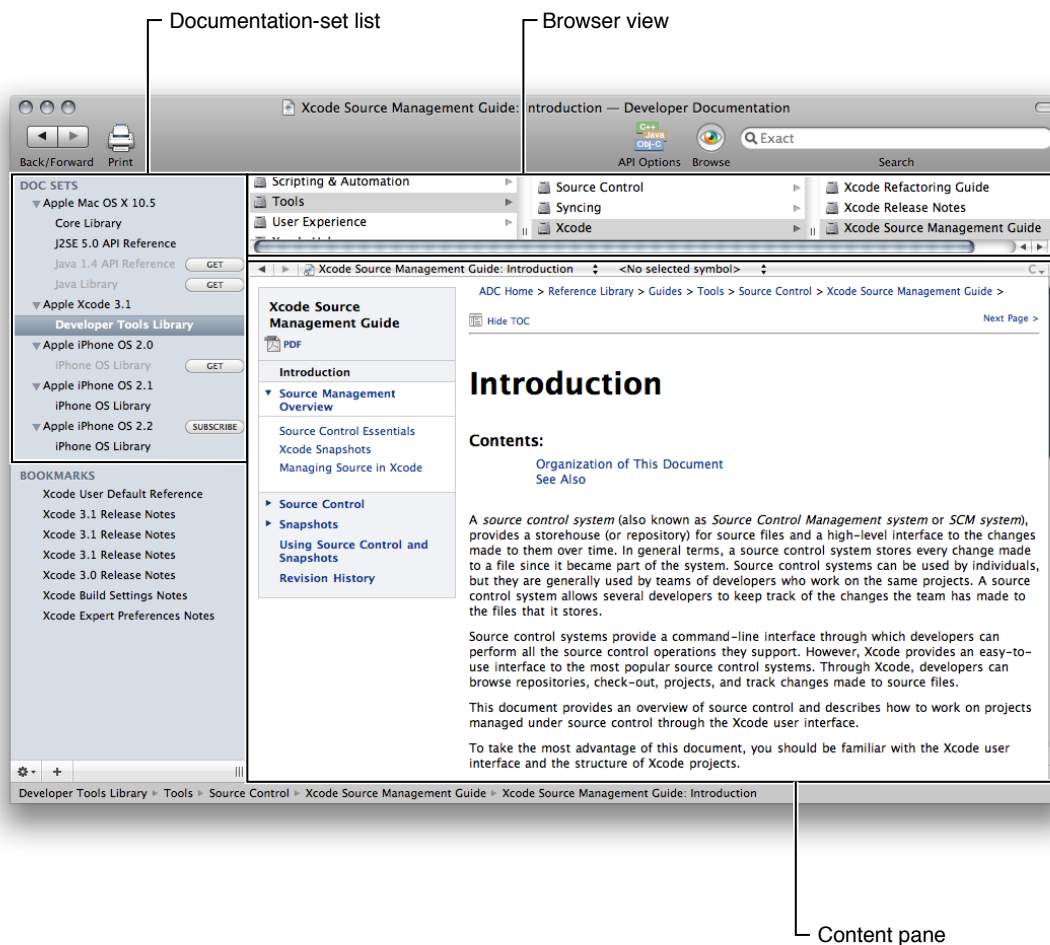
To browse documentation:

1.   In the Documentation window toolbar, click the Browse button so that its icon changes from grayscale to color.

2.   In the documentation-set list, select the doc set you want to browse.

3.   In the browser view, select the categories in which you're interested.

     As you select each category, the content pane shows a page describing the category. When you select a category that doesn't have subcategories, you can select one of the documents that the category encompasses. When you select a document in the browser view, depending on its type and location, it may be displayed in the content pane, your PDF viewer, or your web browser.

Searching Documentation

When you know the subject you're looking for, searching is often the fastest way to find documentation. A documentation search is composed of several components:

- **Search term.** The text that must be present in the documents that make up the search results.

- **Search type.** The Documentation window supports three types of search:

  - **API reference search:** Finds the available reference for a symbol name.

  - **Title search:** Finds the documents whose titles start with, contain, or match the search term.

  - **Full-text search:** Finds the documents whose content matches a search term. The search term can be a word or phrase or may be an elaborate expression using Boolean operators and wildcard characters

- **Search scope.** The doc sets in which Xcode searches for the search term. You can search one doc set or all doc sets.

When defining a documentation search, you use the following Documentation window controls (identified in Figure 5-2):

- **The search field** to specify the search term.

- **The search bar** to specify the search type, search scope, search language (for API reference searches only), and match criterion (for full-text searches only).

- **The documentation-set list** to identify the doc set to be searched (when the search is limited to one doc set).

The set of documents that meet search criteria (the search result) appears in the detail view.

Figure 5-2   Documentation window in search mode
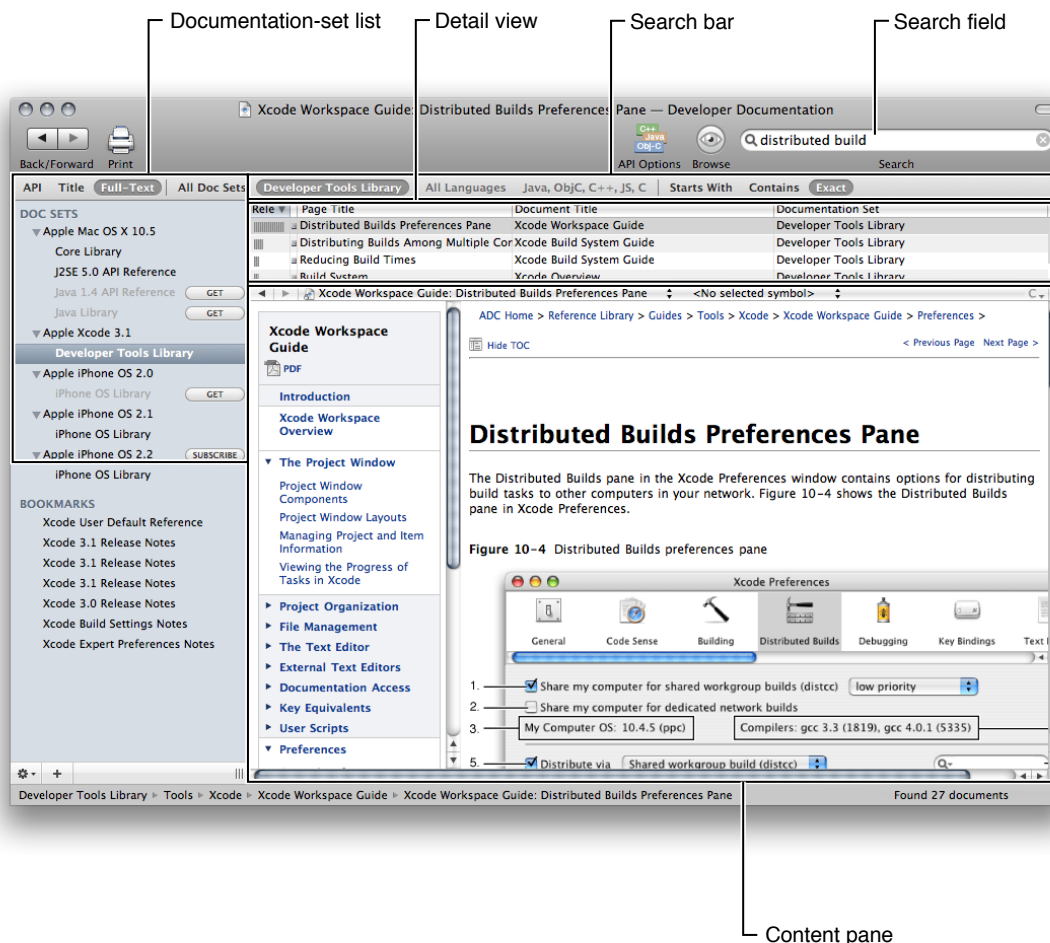
The following sections provide more details about each type of documentation search.

API Reference Search

When you are writing code, you often need to find documentation for a method, function, or data type. The API reference search lets you quickly find the information you need.

To perform an API reference search:

1. In the documentation-set list, select the doc set in which you want search. To search all doc sets, click All Doc Sets in the search bar.

2. In the search bar, click API and specify the programming language you're using.

   You can click the API Options button in the toolbar to configure the list of languages the search bar shows.

3. In the search field, enter the name of the symbol whose reference you want to view.

The detail view displays the documents that match the name entered. Xcode reduces the list of matches as you complete the symbol name.

Title Search

When you are relatively sure of the name of the technology for which you want to find documentation, a title search may provide very relevant results because each of the documents in the search results is guaranteed to address the subject at a useful level.

To perform a title search:

1. In the documentation-set list, select the doc set you want to search. To search all doc sets, click All Doc Sets in the search bar.

2. In the search bar, click Title.

3. In the search field, enter the search term.

    The detail view displays the documents whose titles contain the words in the search term.

Full-Text Search

A full-text search lets you find documentation based on words, terms, and wildcard characters.

To perform a full-text search:

1. In the documentation-set list, select the doc set you want to search. To search all doc sets, click All Doc Sets in the search bar.

2. In the search bar, click Full-Text and click the match criterion you want to use: Starts With, Contains, or Exact:

    ■ **Starts With.** Matches documents that contain words that begin with the words specified in the search term. For example, with the `NS CF` search term, the search result is made up of documents that contain words such as "*NS*Window" and "*CF*String," each document containing at least one word that starts with the string "NS" and one word that starts with the string "CF".

    ■ **Contains.** Matches documents that have words that contain the words specified in the search term. For example, the search term `stringWith UTF` returns documents that contain words such as "*stringWith*Format" and "initWith*UTF*8String," each document containing at least one word that contains the string "stringWith" and one word that contains the string "UTF".

    ■ **Exact.** Matches documents that contain words that exactly match the words specified in the search term, each document containing all the words specified in the search term.

3. In the search field, enter the search term and press Return.

Using Boolean operators, specifying required terms, and using wildcards allow you to define more precise search terms, which produce smaller but more accurate results.

Using Boolean Operators

If you wish to further restrict the parameters of your search, you can combine search conditions using Boolean operators. You can create arbitrarily complex queries to narrow your search to fit your particular criteria.

> **Note:** The smallest unit at which search results are evaluated is a single HTML file; in Apple's developer documentation, this typically corresponds to a section in a chapter, a group of function descriptions, or a class. If your search term is too restrictive, you may not get any results at all.

Xcode supports the following Boolean operators, listed in order of precedence from highest to lowest:

()	logical grouping
!	NOT
&	AND
		OR

For example, to find documents about the Fonts window that deal with underlining or coloring text, you want to find documents that contain the words fonts, window, underline, and color, with "underline" and "color" each grouped with "fonts" and "window." You can do that with the following search term:

```
(fonts window underline) | (fonts window color)
```

Specifying Required Terms

Simpler than a Boolean search term, a required-terms search lets you search for terms that must or must not appear in documents returned as a search result. A required terms search uses the following operators:

+	Indicates a term that must appear in any document returned.
-	Indicates a term that must *not* appear in any document returned.

For example, entering `+window` returns all documents containing the word "window," similar to the behavior you get by simply entering `window` as a search term. However, if you enter `+window -dialog`, you get all documents containing the word "window" but not the word "dialog."

Using Boolean operators to construct the previous search term, you would write:

```
window & (!dialog)
```

Using Wildcards

If you are not sure how a particular term appears in the documentation, you can use a wildcard search to include all variations of a search term in the search results. For example, if you are looking for all documentation about buttons in Mac OS X, you probably really want to see all documentation containing either the word "button" or the word "buttons." Rather than specifying each of these as separate terms, you can simply use the wildcard character to construct the following search term, which returns all documents containing the word "button" or any word with the prefix "button."

```
button*
```

You can use the wildcard character anywhere within a search string. Using a wildcard character at a location other than at the end of a search term may result in longer search times.

Searching API Reference from the Text Editor

There are two ways to search for API reference from the Xcode text editor:

- Option–double-click a symbol name.

- Select an expression and choose Help > Find Selected Text in API Reference.

Both editor-based API reference search mechanisms bring up the Documentation window and, using the current API reference search criteria, perform an API search of the selected symbol.

Using Documentation Bookmarks

The Documentation window also supports bookmarks, to provide easy access to documentation that you use frequently. Xcode includes a default set of bookmarks, but you can delete any of these default bookmarks or create your own.

To add a bookmark, choose Edit > Add to Bookmarks.

This command bookmarks the page currently open in the content view and adds the bookmark to the Bookmarks list. You can also add a bookmark by dragging the document proxy icon in the Documentation window titlebar to the Bookmarks list.

To open a bookmarked location, select the bookmark in the Bookmarks list. To rename or delete a bookmark, use the bookmark's shortcut menu.

Using the Research Assistant

The Research Assistant is a lightweight window that provides a condensed view of the API reference for the selected item (a symbol or build setting) without taking your focus away from the editor in which the item is located. This window provides an unobtrusive way to consult API reference without using the Documentation window. However, when you need to delve deeper into the reference, the Documentation window is just a click away.

To display the Research Assistant, choose Help > Show Research Assistant.

The top part of the Research Assistant (Figure 5-3) contains links to the item's related reference and header file. The rest of the window is made up of several sections, which you can open or close depending on what information you're interested in. These sections include the symbols declaration, abstract, availability, related API, related documents, and sample code.

Figure 5-3   The Research Assistant
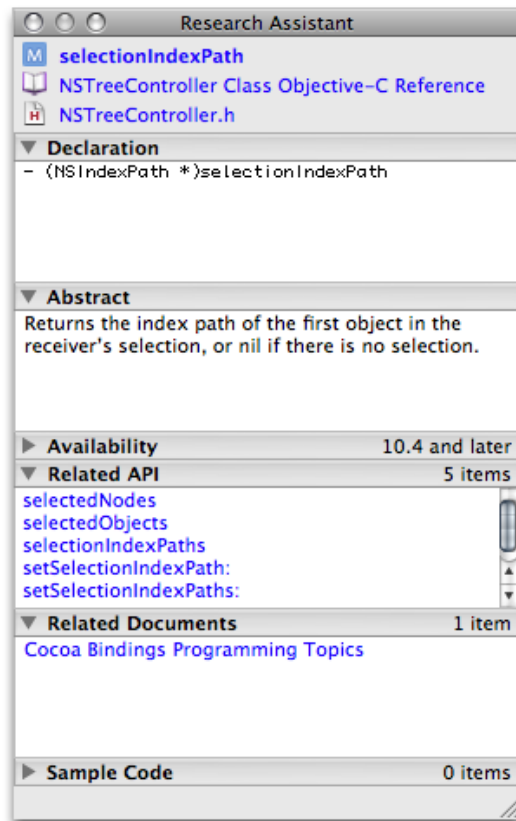
You can use the Research Assistant in the text editor and the build settings editor.

Viewing Man Pages

Online manual (or "man") pages provide reference documentation for BSD and POSIX functions and tools, as well as command-line tools such as `xcodebuild`. You can find man pages in Xcode in either of the following ways:

- Type the name of the tool or function into the search field of the Documentation window. Xcode includes the man page entries for standard C and C++ system calls in its API reference search and includes all man page entries in its full-text search. Note that you must have C language searching enabled to find man pages for system calls using the API reference search.

- Choose Help > Open man page.

  Use the "man page name" option to display documentation on a command-line tool. You can optionally specify a man page section—for example, `access(5)`. Use the "search string" option to find commands that are related to a keyword.

Subscribing to Documentation Feeds

Xcode uses documentation sets as the source of the content is displays in the Documentation window. The entities (organizations or individuals) that create documentation sets are known as documentation-set publishers. Apple is one such publisher, but other entities can create and publish their own doc sets. They usually make them available to developers through documentation feeds. A **documentation feed** is an RSS-style web feed that Xcode can check periodically to determine when a publisher makes available updates to a doc set or releases a new doc set.

If you want Xcode to let you know when a publisher whose doc sets you are interested in releases new content, subscribe to that publisher's documentation feed. To subscribe to a documentation feed, click the Subscribe button next to the publisher's (or feed) name in the documentation-set list. When you don't have any of a publisher's doc sets installed on your computer, create a subscription by choosing New Subscription from the action menu in the bottom-left corner of the Documentation window. In the dialog that appears, enter the feed's URL, which you obtain from the publisher.

When a new doc set becomes available, it's listed under the feed name with a Get button next to it. Click the button to download and install the doc set.

When a new release of a doc set you have on your computer is available, an Update button appears next to the doc set's name. Click the button to download and install the new version of the doc set.

> **Note:** If you do not subscribe to a documentation feed, Xcode does not notify you of documentation updates the publisher of that feed makes.

Documentation Preferences

The Documentation pane of Xcode preferences (Figure 5-4) provides options for specifying the location of doc sets in your file system and setting the minimum font size used in the content pane in the Documentation window.

Figure 5-4 Documentation preferences pane
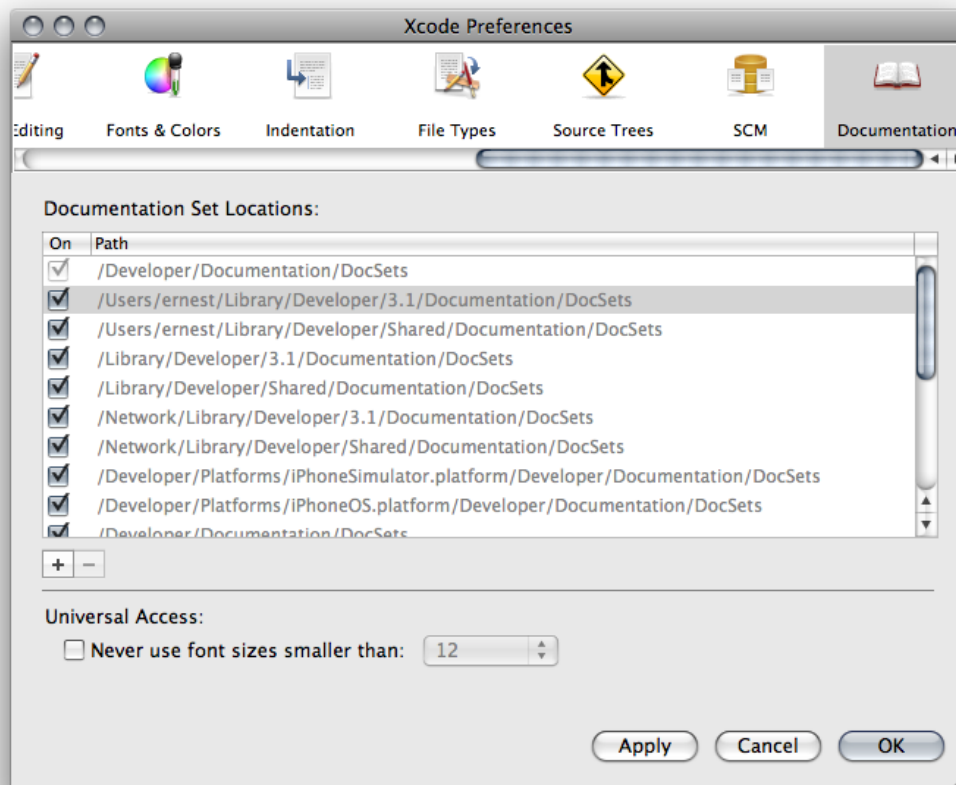
Here's what the pane contains:

■ **Documentation Set Locations.** This list identifies the directories that contain documentation sets. You can customize this list to relocate documentation sets or if you want to share documentation sets between multiple Xcode releases installed on your computer. (You must manually move each doc set to its new location.) Note that for each Xcode release you cannot relocate the Developer Tools Library documentation set.

■ **Universal Access.**

❏ **"Never use font sizes smaller than".** Specifies the minimum font size to use in the content pane of the Documentation window.

Keyboard Shortcuts

Xcode lets you change the keyboard shortcuts for actions accessible through menu items or the keyboard, such as paging through a document or moving the cursor. You can choose a predefined set of keyboard shortcuts for menu items and other tasks, or you can create your own set. The predefined sets include sets that mimic BBEdit, Metrowerks, CodeWarrior, and MPW.

This chapter shows how to view and change the keyboard shortcuts for menu items and key-based actions.

Key Bindings Preferences

The Key Bindings pane in Xcode preferences (Figure 6-1) lets you see and customize the list of Xcode commands and their keyboard shortcuts.

Figure 6-1   Key Bindings preferences
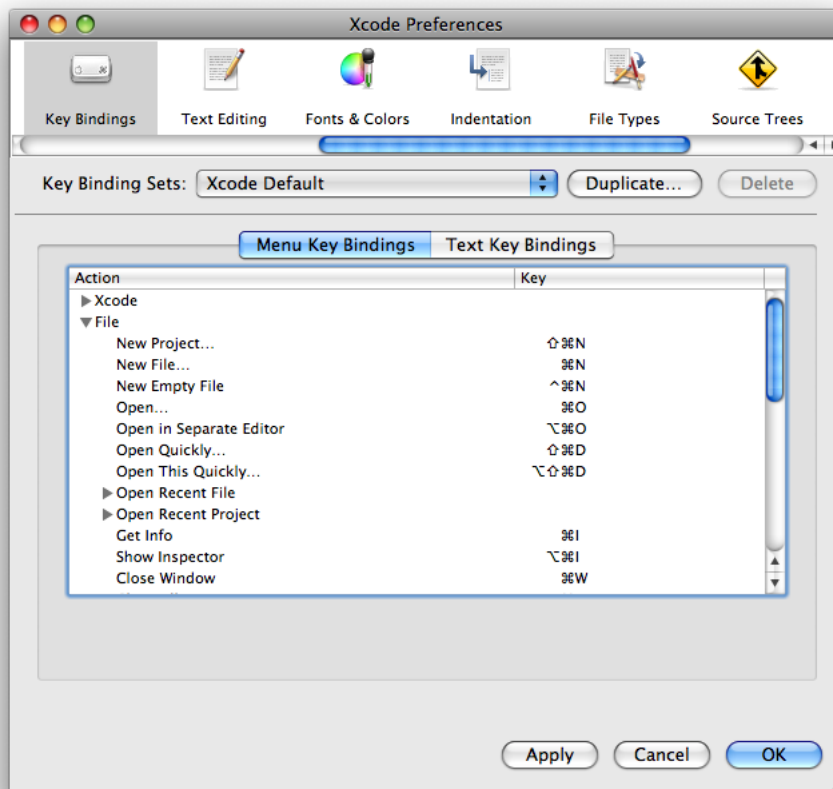
Here is what the pane contains:

■ **Key Bindings Sets menu.** This menu lets you choose which set of key bindings are in effect. Xcode provides four predefined sets: Xcode Default, BBEdit Compatible, Metrowerks Compatible, and MPW Compatible. You can also add your own custom sets of key bindings.

■ **Duplicate button.** You cannot edit any of the built-in key bindings sets. To create your own set of custom key bindings, click this button to create a copy of the current set and edit that copy.

■ **Delete button.** This button deletes a custom key binding set.

■ **Menu Key Bindings pane.** This pane lists the key bindings for menu items in Xcode. See "Customizing Keyboard Shortcuts for Menu Items" (page 100) for details.

■ **Text Key Bindings pane.** This pane lists the key bindings for text editing actions in Xcode's editor. See "Customizing Keyboard Shortcuts for Other Tasks" (page 101) for details.

■ **Action/key list.** The Menu Key Bindings and the Text Key Bindings pane contain an action/key list, which list the actions available in Xcode and their corresponding keyboard shortcuts. The Action column lists the Xcode action—a menu item or text editing command—and the Key column lists the keyboard shortcut for that action. To edit the key binding for a command, double-click in the Key column and type the key combination. You can assign more than one keyboard shortcut to an action. To add additional key combinations, click the plus (+) button.

Customizing Keyboard Shortcuts for Menu Items

The Menu Key Bindings pane in Key Bindings preferences (shown in Figure 6-1 (page 99)) provides access to most of the menus and menu items in Xcode.

Xcode lists keyboard shortcuts using the traditional menu glyphs shown in Figure 6-2 (not all glyphs are shown).

Figure 6-2   Some of the glyphs that represent keys

Command	⌘	Shift	⇧
Option	⌥	Control	⌃
Left Arrow	←	Home	↖
Right arrow	→	End	↘
Up arrow	↑	Page up	⇞
Down arrow	↓	Page down	⇟
Backspace	⌫	Return	↩
Delete	⌦	Enter	⌤
Escape	⎋		

The following steps describe how to create a custom set of key bindings, based on the Xcode Default set, and how to add a keyboard shortcut. In this example, the keyboard shortcut performs a full-text documentation search.

1. Open Key Bindings preferences.

2. From the Key Binding Sets menu, choose "Xcode Default."

3. Click the Duplicate button to create a copy of that set. When prompted for a name for the set, type `My Set`.

4. Click Menu Key Bindings.

5. In the Action list, scroll to Help and open that section.

6. Double-click in the Key column in the Find Selected Text in Documentation row to open an editing field, then press Shift-Command-T (holding the keys down simultaneously). The result is shown in Figure 6-3.

    If you try to assign a keyboard shortcut that is already assigned to another action in the current key binding set, Xcode displays a message indicating which action it is assigned to below the key bindings table.

    Note that Xcode displays the letter "t" in its capitalized form. Whether or not you include the Shift key as part of a menu keyboard shortcut, Xcode shows letters as they appear in menus (that is, as capitals).

    You can use the minus (-) button to clear a menu keyboard shortcut. You can use the plus (+) button to assign multiple shortcuts to a single action (and you can use any of the shortcuts to initiate the action).

Figure 6-3   Editing a keyboard shortcut for a menu item
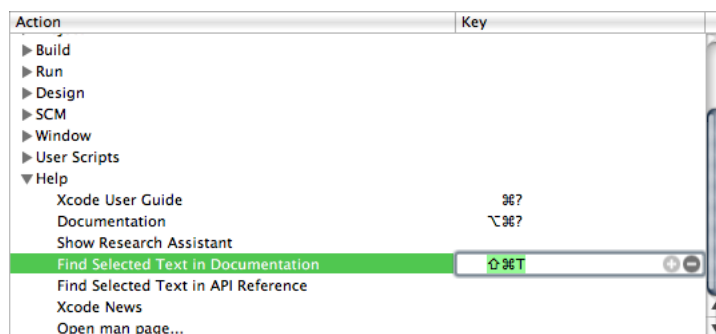
7. You can repeat the previous step to add or change other menu keyboard shortcuts.

8. Click Apply or OK to apply your changes.

9. You can now press Shift-Command-T to perform a full-text search of the documentation for the selected text.

Customizing Keyboard Shortcuts for Other Tasks

You can customize keyboard shortcuts for tasks such as editing and formatting text, cursor movement, and project navigation using steps similar to those described for menu items in "Customizing Keyboard Shortcuts for Menu Items" (page 100). In addition to its default settings, Xcode provides sets that are compatible with BBEdit, Metrowerks CodeWarrior, and MPW.

The following steps show how to set a shortcut for the Capitalize Word action:

1. Open Key Bindings preferences.

2. From the Key Binding Sets menu, choose My Set.

   If My Set is not available, create it as described "Customizing Keyboard Shortcuts for Menu Items" (page 100)).

3. Click Text Key Bindings.

4. Double-click in the Keys column in the Capitalize Word row to open an editing field, then press Shift-Control-C (holding the keys down simultaneously). The result is shown in Figure 6-4.

   Figure 6-4 Editing a keyboard shortcut for an editing action
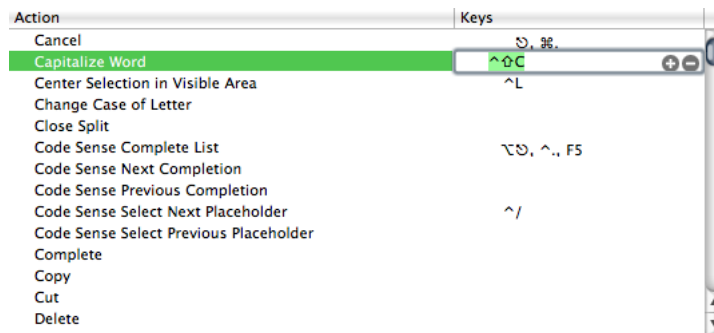
   You can use the minus (-) button to clear a keyboard shortcut. You can use the plus (+) button to assign multiple shortcuts to a single action (and you can use any of the shortcuts to initiate the action).

5. You can repeat step 4 to add or change other keyboard shortcuts for editing actions (or other actions not shown here).

6. Click Apply or OK to apply your changes.

You can now press Shift-Control-C to capitalize the currently selected word in an editable text control.

User Scripts

User scripts are shell scripts you can define in your workspace and that you can execute by choosing them from the User Scripts menu or using a keyboard shortcut. With these scripts you can invoke command-line tools and perform operations on the selected text. For example, you can sort the selected lines in the text editor.

Xcode provides a number of mechanisms for working with user scripts. These features include:

■   The ability to execute text in a text editor window as a shell command or series of commands

■   A number of built-in script variables and utility scripts you can use in menu scripts or other scripts

This chapter describes how to manage and use Xcode user scripts.

Managing User Scripts

Xcode provides predefined user scripts that let you open files, perform searches, add comments to your code, sort text, and even add HeaderDoc templates that can help you document your header files. And you can use these files as examples to help write custom scripts.

The User Scripts menu (Figure 7-1) shows the predefined user scripts and the user scripts you add. This menu reflects the user-script hierarchy defined in the user-scripts editor, shown in Figure 7-2.
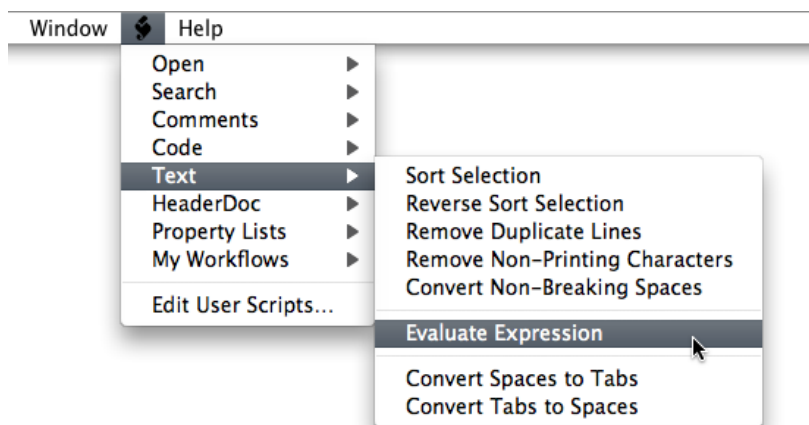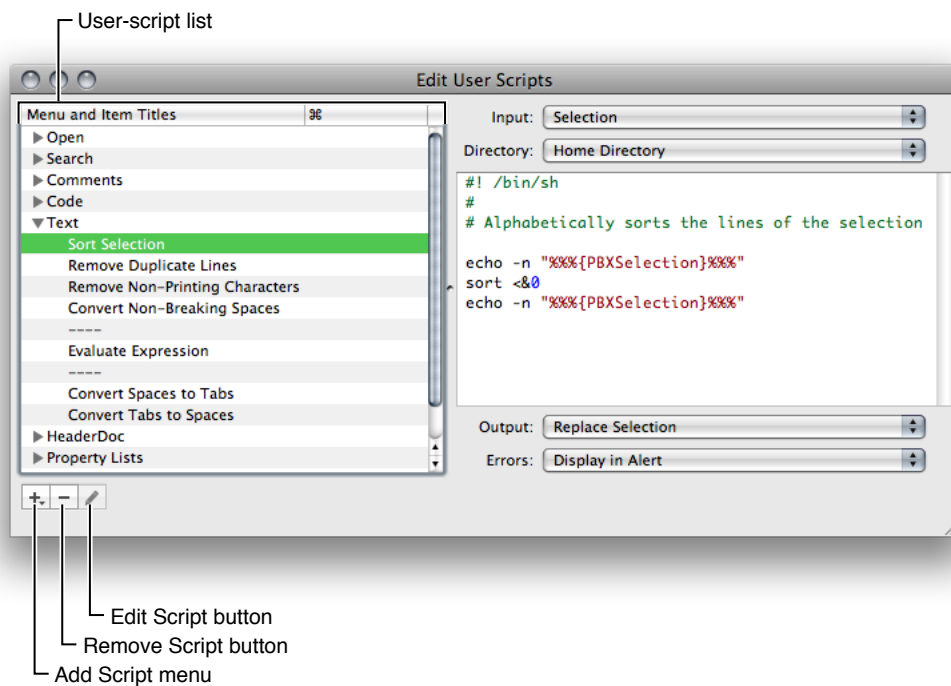
Figure 7-1   User Scripts menu

Figure 7-2   User-scripts editor

You use the user-scripts editor to add, modify, and delete user scripts. You can also add user-script groups (menu groups) and separators.

Adding a user script adds a corresponding menu item to the User Scripts menu. You can define the script in the text editor pane of the user-scripts editor, or you can specify the location of an existing shell script in your file system.

A user script has four attributes, whose values you specify in the user-scripts editor:

- **Input.** The script's input. It can be nothing, the selection, or the current file.

- **Directory.** The script's initial working directory. It can be your home directory, the path indicated by the selected text, or the file system root directory (/).

- **Output.** The script's output mechanism. It can be none, a dialog, the Clipboard, and others.

- **Errors.** The destination of the script's error messages. It can be none, a dialog, the Clipboard, or the script's regular output mechanism.

To add a user script:

1. Select the user-script group into which you want to add the user script.

2. Choose User Scripts > Edit User Scripts to open the user-scripts editor.

3. Click the Add Script button (+) and choose one of the following items from the menu:

   - **New Shell Script.** Creates a user-script menu item and a blank shell script, which you edit in the editor pane.

- **Add Script File.** Creates a user-script menu item and prompts you for the location of an existing shell script file.

- **Add Automator Workflow.** Creates a user-script menu item and prompts you for the location of an existing Automator workflow.

4. Enter the name and keyboard shortcut (if desired) for the user script in the user-script list, as shown in Figure 7-3.

Figure 7-3   Adding a shell script
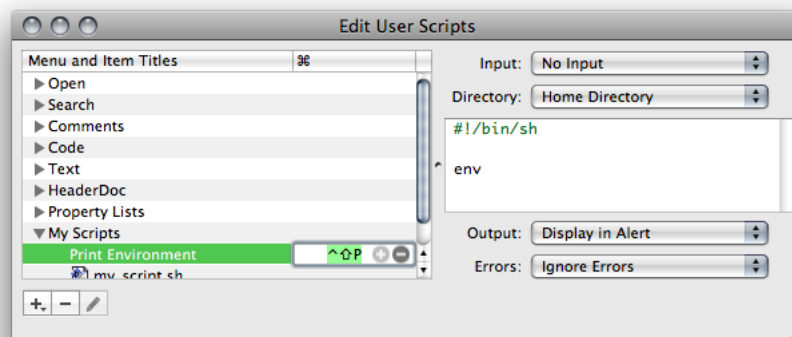
5. Choose values from the Input, Directory, Output, and Errors menus.

To remove a user script, user-script group, or separator, select it and click the Remove Script button.

To edit a file-based user script, select the script in the user-scripts list and click the Edit Script button (this button is available only for file-based user scripts).

Duplicating User Scripts

Sometimes it may be more convenient to create a user script based on an existing one instead of writing one from scratch. To create a custom user script based on a predefined user script:

1. Choose User Scripts > Edit User Scripts to open the user-scripts editor.

2. Select the user script to duplicate.

3. Click the Add Script button and choose Duplicate Script from the menu.

4. Customize the script and its attributes as appropriate.

Advanced User Scripts

The following sections describe advanced features of shell-based user scripts. The topics covered include variables you can use in menu script definitions, variables expanded prior to script execution, and special user output script markers.

Script Input Variables

Shell-based user scripts can also contain a variety of special variables that are expanded by Xcode each time the script is executed. The following sections describe the supported variables.

Getting Text from the Active Window

These variables are replaced by text in the active window:

- `%%%{PBXSelectedText}%%%` is replaced by the selected text in the active text object.
- `%%%{PBXAllText}%%%` is replaced by the entire text in the active text object.

The text is expanded verbatim with no quotation marks. In most shells this would be a dangerous practice because the selection might include single or double quotation marks or any number of other special shell characters. One safe way to use this in a Bourne shell script, for example, is to have it expand within "here-doc" style input redirection like so:

```
cat << EOFEOFEOF
%%%{PBXSelectedText}%%%
EOFEOFEOF
```

The script above would simply print the selected text to the standard output.

Getting Information on the Contents of the Active Window

These variables are replaced by information on the text in the active window:

- `%%%{PBXTextLength}%%%` is replaced by the number of characters in the active text object.
- `%%%{PBXSelectionStart}%%%` is replaced by the index of the first character in the selection in the active text object.
- `%%%{PBXSelectionEnd}%%%` is replaced by the index of the first character after the selection in the active text object.
- `%%%{PBXSelectionLength}%%%` is replaced by the number of characters in the current selection in the active text object.

Getting the Pathname for the File in the Active Window

This variable Is replaced by the path to the file for the active text object if it can be determined:

```
%%%{PBXFilePath}%%%
```

This result may not be accurate. Xcode tries to find the file path first by walking up the responder chain looking for a window controller that has a document. If it finds one it uses the document's filename. If it does not find one, it uses name of the window's represented file. For more information, see `NSWindowController` and `NSDocument`.

Sometimes this variable expands to nothing and sometimes it may expand to a filename that is not really a text file containing the text of the active text object. In the Xcode text editor this works correctly. In other text areas in Xcode (like the build log or any text field) it does not do anything reasonable.

Getting the Pathname for the Utility Scripts

This variable is replaced by the path to the folder that contains a number of built-in utility scripts and commands:

```
%%%{PBXUtilityScriptsPath}%%%
```

These scripts and commands can be used from user scripts to provide functionality such as presenting a dialog to ask the user for a string or to ask the user to choose a folder or file, or to add to the menu bar of the host application. See "Built-in Utility Scripts" (page 108) for descriptions of the available utility scripts.

Script Output Markers

When a user script is done executing, Xcode scans the output for certain special markers. Only one output marker is supported. This variable specifies an exact selection within the output:

```
%%%{PBXSelection}%%%
```

By default, Xcode sets the selection to be an insertion point after the newly inserted output text. But if the output contains one or two instances of this special marker, it uses them to determine the selection. If there is one such marker, it identifies an insertion point selection. If there are two, all the text between them is selected. Xcode then removes the markers from the output.

Using Utility Scripts

Xcode provides several useful built-in utility scripts, which you can use in the user scripts you create.

To use one of these scripts, preface it with the expansion variable `%%%{PBXUtilityScriptsPath}%%%`, which specifies the location of the script. For example, the following statement displays a dialog to get input from the user and places the result in the variable `STRING`. The original text displayed in the dialog is "DefaultString".

```
STRING = `%%%{PBXUtilityScriptsPath}%%%/AskUserForStringDialog "DefaultString"`
```

In addition to the `AskUserForStringDialog` script, Xcode provides built-in scripts to:

■   Choosing a new file

■   Choose an existing file or folder

■   Choose an application

For details, see "Built-in Utility Scripts" (page 108).

Built-in Utility Scripts

Xcode provides several useful utility scripts that are built in to Xcode itself. These scripts can be used in menu definition file scripts or in MPW-style worksheet content. To use one of these scripts, use the `%%%{PBXUtilityScriptsPath}%%%` expansion variables. For an example, see "Using Utility Scripts" (page 107).

Specifying a String

`AskUserForStringDialog [`*`default-string`*`]`

Displays a dialog in the active application and returns the string that the user enters. If supplied, *default-string* is the initial content of the text field.

Choosing an Existing File or Folder

`AskUserForExistingFileDialog [`*`prompt-string`*`]`

`AskUserForExistingFolderDialog [`*`prompt-string`*`]`

Displays a standard open dialog and returns the path of the file or folder that the user chooses. If supplied, *prompt-string* is the prompt in the dialog. Otherwise a default prompt is used.

Choosing a New File

`AskUserForNewFileDialog [`*`prompt-string`* `[`*`default-name`*`]]`

Displays a standard save dialog and returns the path of the new file. If supplied, *prompt-string* is the prompt in the dialog. Otherwise, a default prompt is used. If supplied, *default-name* is the default name for the new file.

Choosing an Application

`AskUserForApplicationDialog [`*`title-string`* `[`*`prompt-string`*`]]`

Displays an application picker dialog and returns the path of the application the user chose. If supplied, *title-string* is the title for the dialog. Otherwise, a default title is used. If supplied, *prompt-string* is the prompt in the dialog. Otherwise, a default prompt is used.

Adding a Menu Item from Any Script File

`SetMenu add script` *`script menu-title key-equiv input-treatment output-treatment index`* `[`*`menu-path`* `...]`

Adds a menu item to an existing menu in Xcode. The menu item has the name *menu-title*, and the keyboard shortcut *key-equiv*. (Use `""` for no keyboard shortcut.) When the user chooses this command, it invokes the script *script*, getting its input from *input-treatment* and placing its output in *output-treatment*. The new item is inserted at *index* in the menu identified by *menu-path*. If you don't specify *menu-path* the item appears in the menu bar. *menu-path* contains the titles of menus and submenus that lead to the desired menu.

index is a zero-based index starting at the end of all the original items in the menu. For example, the index `0` in the File menu would generally be the first item after Print (usually the last item in the File menu of an application). Index `2` would be after the second custom item in a menu. Use negative indices to count from the end of a menu. Index `-1` means "at the end," and Index `-2` means "right before the last item."

The *key-equiv*, *input-treatment*, and *output-treatment* arguments use the same syntax as the values of the menu definition file directives `PBXKeyEquivalent`, `PBXInput`, and `PBXOutput` respectively. For example, if *input-treatment* is `Selection`, the selected text is the input for the new menu item's script.

This is the most complicated form of the `SetMenu` command. Usually it is better to use the form described in"Adding a Menu Item from a Menu Definition Script" (page 109) in conjunction with menu script definition files.

Adding a Menu Item from a Menu Definition Script

```
SetMenu add scriptfile script-pathindex [menu-path ...]
```

Adds menu items to an existing menu in Xcode. The items are read from *script-path*. See "Adding a Menu Item from Any Script File" (page 108) for details on the file format. Details such as the menu items keyboard shortcuts, and input and output treatment are defined within the file. The new items are inserted at the given *index* in the menu specified by *menu-path*. If you don't specify *menu-path*, the item appears in the main menu bar. *menu-path* contains the titles of menus and submenus that lead to the desired menu.

index is a zero-based index starting at the end of all the original items in the menu. For example, the index `0` in the File menu would generally be the first item after the Print command (usually the last item in the File menu of an application). Index `2` would be after the second custom item in a menu. Use negative indices to count from the end of a menu. Index `-1` means "at the end," and Index `-2` means "right before the last item."

Adding a Submenu

```
SetMenu add submenu submenu-nameindex [menu-path ...]
```

Adds a submenu to an existing menu in Xcode. The submenu's title is *submenu-name*. Initially, it has no items. The new submenu is inserted at *index* in the menu specified by *menu-path*. If you don't specify *menu-path*, the item appears in the main menu bar. *menu-path* contains the titles of menus and submenus that lead to the desired menu.

index is a zero-based index starting at the end of all the original items in the menu. For example, the index `0` in the File menu would generally be the first item after the Print command (usually the last item in the File menu of an application). Index `2` would be after the second custom item in a menu. Use negative indices to count from the end of a menu. Index `-1` means "at the end," and Index `-2` means "right before the last item."

Adding a Menu Separator

```
SetMenu add separator index [menu-path ...]
```

Adds a separator to an existing menu in Xcode. The new separator is inserted at *index* in the menu specified by *menu-path*. If you don't specify *menu-path*, the item appears in the main menu bar. *menu-path* contains the titles of menus and submenus that lead to the desired menu.

index is a zero-based index starting at the end of all the original items in the menu. For example, the index 0 in the File menu would generally be the first item after the Print command (usually the last item in the File menu of an application). Index 2 would be after the second custom item in a menu. Use negative indices to count from the end of a menu. Index -1 means "at the end," and Index -2 means "right before the last item."

Removing a Custom Menu Item

```
SetMenu remove item index [menu-path ...]
```

Removes a custom item from an existing menu in Xcode. The custom item at *index* in the menu specified by *menu-path*. If you don't specify *menu-path*, the item is removed from the main menu bar. *menu-path* contains the titles of menus and submenus that lead to the desired menu.

index is a zero-based index starting at the end of all the original items in the menu. For example, the index 0 in the File menu would generally be the first item after the Print command (usually the last item in the File menu of an application). Index 2 would be after the second custom item in a menu. Use negative indices to count from the end of a menu. Index -1 means "at the end," and Index -2 means "right before the last item."

Only items and submenus added by a `SetMenu` command can be removed by the `SetMenu remove item` command. You cannot remove the standard Xcode menu items.

Removing All Custom Menu Items from a Menu

```
SetMenu remove all [menu-path ...]
```

Removes all custom items from an existing menu in Xcode. All custom items in the menu identified by *menu-path* are removed. If you don't specify *menu-path*, the item is removed from the main menu bar. *menu-path* contains the titles of menus and submenus that lead to the desired menu.

This command applies to items and custom submenus but does not recurse into original submenus. For example, if you add an item to the File menu and you add a menu called My Scripts to the main menu bar, `SetMenu remove all` removes the My Scripts menu but does not remove the custom item in the File menu. `SetMenu remove all File` removes the custom item from the File menu.

Only items and submenus added by a `SetMenu` command can be removed by the `SetMenu remove all` command. You cannot remove the standard Xcode menu items.

Resetting Xcode

To reset Xcode to its factory settings for the logged-in user, run these commands in Terminal:

```
> defaults delete com.apple.Xcode
> rm -rf ~/Library/Application\ Support/Xcode
```

Glossary

browse mode A mode of viewing documentation in the Documentation window that lets you browse the documents in a documentation set by navigating its category hierarchy. See also **search mode**.

code completion A text editor feature that helps you to type less by offering completions to the source code you type.

code focus A text editor feature that highlights a source file's scope levels using a grayscale.

code folding A feature that allows you to hide code in the text editor.

Code Sense A technology used to index source code. The generated indexes are used by code completion, API reference lookup, and refactoring.

completion list A list of completions based on an entered token.

documentation feed An RSS-style web feed that Xcode can check periodically to determine when a documentation-set publisher makes available updates to a doc set or releases a new doc set.

documentation set (doc set) A structured directory containing HTML-based content as well as indexes into that content. You view doc-set content in the Documentation window.

Documentation window A window that lets you browse and search the developer documentation installed on your computer. See also **documentation set**.

file reference A pointer to a file used in the Groups & Files list.

group An item in the Groups & Files list that encloses a set of project items. See also **static group**, **smart group**.

File History menu A pop-up menu in the navigation bar that lists the files opened in a text editor window.

focus box A box that code focus uses to identify scopes. See **code focus**.

focus center The current scope level, demarcated by the focus box. See **code focus**.

focus ribbon A control that lets you survey the the scope of code. It uses a grayscale to identify the scope level of the corresponding code lines. See **code focus**.

Function menu A pop-up menu in the navigation bar that lets you navigate a source file.

Groups & Files list A list in the project window that provides an outline view of a project's contents.

gutter The area in the left side of the text editor that displays line numbers, identifies code lines with warnings or errors, and lets you manage breakpoints.

message bubble A view that displays a project message or breakpoint in the text editor, adjacent to the code line to which the information applies.

navigation bar A bar at the top of the a text editor that allows you to navigate through a file or its related files, go through the files you've opened in that editor, list breakpoints in the current file, and more.

nib file A file that stores a graph of user interface objects that define your product's user interface.

project window A window that displays and organizes a project's source files, targets, and executables.

Research Assistant A lightweight window that provides a condensed view of the API reference and links to related documentation for the selected symbol or build setting without taking the focus away from the editor in which the item is located.

search mode A mode of viewing documentation in the Documentation window that lets you search for documentation using variety of criteria. See also **browse mode**.

smart group A group in the Groups & Files list that contains built products, search results, bookmarks, project symbols, as well as source files. These groups are populated automatically. See also **static group**.

static group A group in the Groups & Files list that contains source files, such as source code files, header files, implementation files, frameworks, nib files, and other files. These groups are prepopulated through project templates or modified manually. See also **smart group**.

status bar A bar in Xcode windows that displays status messages about your project, such as whether a build was successful.

target A project item that contains the instructions for building a software component or product.

text macro A text editor shortcut for entering source code.

token A string containing no space characters.

Document Revision History

This table describes the changes to *Xcode Workspace Guide*.

Date	Notes
2009-01-06	Made minor content fixes.
	Added steps to reset Xcode to its factory settings in "Resetting Xcode" (page 111).
2008-11-14	Updated screen shots and incorporated editorial improvements.
2008-09-09	Added missing cross-references.
2008-05-22	New document that provides an overview of the Xcode workspace, and shows how to use its components and features.
	Updated "User Scripts" (page 103) for Xcode 3.0.