A Tour of Xcode

Tools & Languages: IDEs

2009-07-22

Contents

Figures and Listings

Introduction

> **Note:** This document was previously titled *Xcode Quick Tour for Mac OS X*.

The Xcode toolset is Apple's integrated suite of software development tools that includes compilers and applications, together with an extensive set of programming libraries and interfaces. The centerpiece of these tools is the Xcode application, which provides an elegant, powerful user interface for creating and managing software development projects. (Elsewhere in this document, the name Xcode usually refers to the Xcode application.)

If you're new to using Xcode and you want to develop applications for Apple products, reading this document is a good way to get started. This document gives you an overview of the main features in Xcode and a hands-on introduction to creating an application. To get the most out of this tutorial, you should already be familiar with C programming and the Mac OS X user interface. You don't need any previous experience developing software for Apple products.

To follow the instructions in this document, you must install Xcode on your computer. To download the latest version of the Xcode installer package, visit the ADC Developer Tools website.

> **Important:** This document is targeted for Mac OS X v10.6 or later, and Xcode 3.2 or later. Before continuing, make sure your development environment meets these requirements. To find out which version of Xcode is installed on your computer, open the Xcode application (`/Developer/Applications`) and choose Xcode > About Xcode.

Organization of This Document

This document contains the following chapters:

- "Xcode Features Overview" (page 9) explores the major features by looking at an existing Xcode project. This chapter will help you master Xcode terminology and get familiar with Xcode's layout so that you'll be ready to create your own project.

- "Xcode Workflow Tutorial" (page 17) shows how to create a Cocoa version of a Hello World application. The tutorial is designed to show you the workflow that you'll use for creating your own software—from creating a project to getting an application to run. You'll also see how to fix compile-time errors and get an introduction to the debugger.

- "Recommended Reading for Xcode Developers" (page 31) points out some of the most important resources for each phase of development—from the design of your software product to its delivery.

- "Finding and Viewing Documentation" (page 35) provides an overview of the ways you can access, search, and view Apple's developer documentation from within Xcode.

See Also

After reading this document, you may want to take a look at the Xcode documentation set that's available from within Xcode by choosing Help > Xcode Help.

Xcode Features Overview

Xcode is a highly customizable integrated development environment (IDE) with many features that let you create a pleasant and efficient working environment. This overview introduces you to Xcode features by looking at an existing Xcode project—*CFLocalServer*, which builds client and server programs using UNIX domain sockets. Don't worry—you don't need to know anything about UNIX. You'll be looking at this project just to learn the layout of an Xcode project. This project builds more than one product, which is why it's an interesting one to look at.

Before you get started, download *CFLocalServer* from the ADC Mac OS X Reference Library. Double-click the `CFLocalServer.xcodeproj` file to open the project in Xcode.

The Project Window

Every software product starts out as a project. A project is the repository for all the elements used to design and build your product—including source files, user interface specifications, sounds, images, and links to supporting frameworks and libraries. The most visible type of software product you can create with Xcode is an application, but that's not the only kind of project you can create. You can also develop Automator actions, command-line tools, frameworks, plug-ins, and kernel extensions.
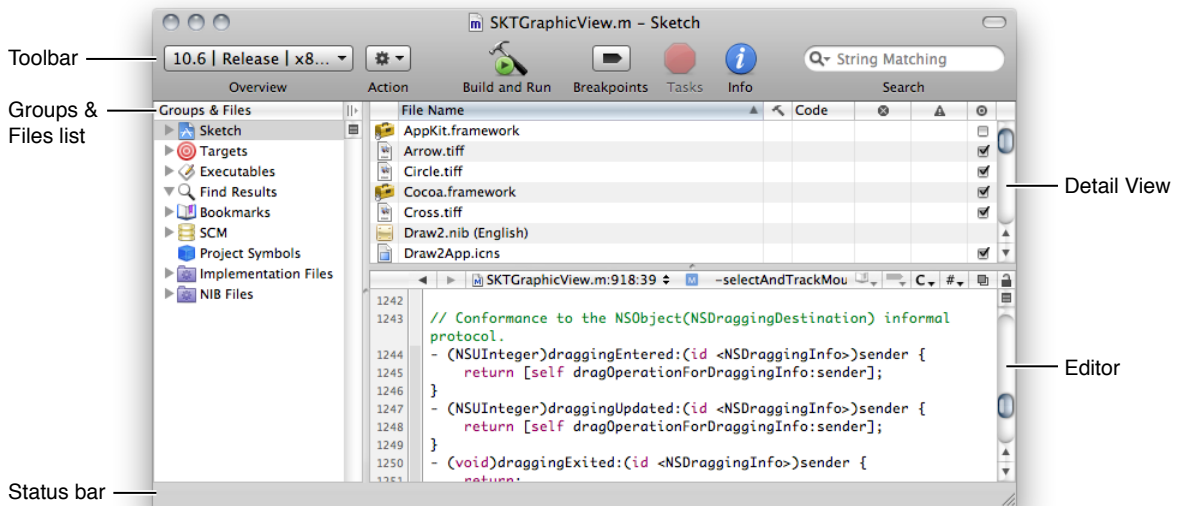
The project window is the control center for an Xcode project. This section briefly describes each of the components in this window, except for the editing pane, which is described in "Text Editor" (page 11).

The Toolbar

The project window toolbar (identified in contains buttons and other controls you can use to perform common operations.
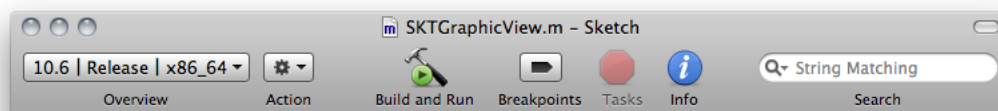
- The Overview pop-up menu lets you choose the active software development kit (SDK), configuration, target, executable, and architecture.

- The Action pop-up menu lists tasks you can perform on the currently selected item. You get the same menu when you Control-click an item.

- The Build and Run button compiles, links, and launches your application.

- The Breakpoints button turns on any breakpoints that you've set and changes the Build and Run button to Build and Debug.

- The Tasks button allows you to stop any operation in progress.

- The Info button opens a window that displays information and settings for groups, files, and targets in your project. You can change many settings in an Info window.

- The search text field filters the items currently displayed in the detail view.

You can customize what's available on the toolbar.

Try it: Customize the toolbar.

1. Control-click the toolbar and choose Customize Toolbar from the pop-up menu.

2. Drag the Run/Debug icon from the sheet that appears to the space in the toolbar located between the Build and Run button and the Breakpoints button.

Groups & Files

Groups organize the files and information in a project—they don't necessarily reflect the actual structure of the project directory or the way the build system views the files. Their purpose is to help you organize your project and quickly find project components and information. You can customize some of the default groups in the list and define groups of your own.

Xcode provides several built-in groups such as Targets, Executables, Bookmarks, and SCM (source control management). The first group listed is always the project group, which organizes all the components needed to build your product. The project group typically contains subgroups for your project's source files, resource files, frameworks, and products.

To the right of the Groups & Files list is the detail view. The detail view is a flattened list of all the items that are currently selected in the Groups & Files list. You can quickly search and sort the items in the detail view, gaining rapid access to important information in your project.

> **Try it:** Explore the groups in the CFLocalServer project.
>
> ■ To see the contents of a group, select the group or click its disclosure triangle.
>
> ■ To create a new group, choose Project > New Group.
>
> ■ To create a new group that automatically finds and organize its contents by a filter or regular expression, choose Project > New Smart Group.

Text Editor

In addition to displaying content that you can edit, the text editor provides a navigation bar at the top, and a gutter and focus ribbon to the left of the content. You can view and edit a source file within the project window by selecting the file in the Groups & Files list. If you prefer to view and edit a source file in its own window, double-click the file in the Groups & Files list.
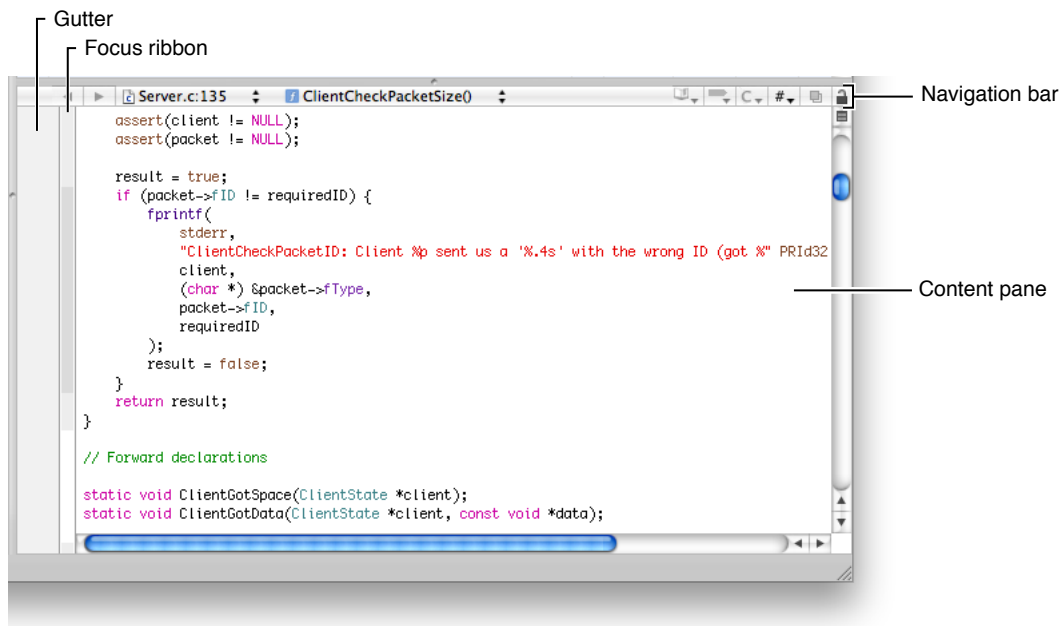
The gutter displays line numbers, as well as information about the location of breakpoints, errors, or warnings in the file.

The focus ribbon, as its name suggests, helps you to concentrate your attention on parts of your code by:

■ Identifying the scope of a block of code

■ Allowing you to hide and show blocks of code

The bar along the top of the editor contains several menus and buttons that let you move between files you've viewed, jump to symbols, and switch to other open files.

Figure 1-1   Text editor navigation bar
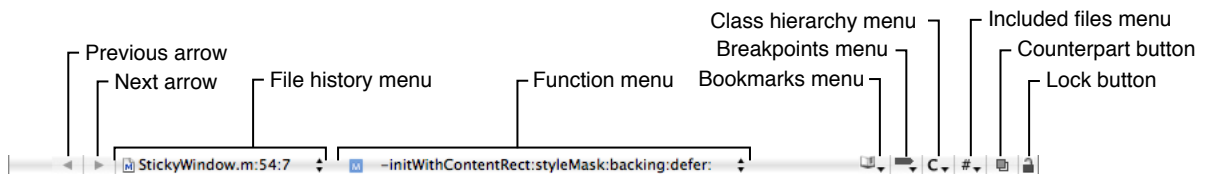
Although you can hold the pointer over each item to see a tooltip that provides a description of each item, these items need more explanation:

- The 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. The menu lets you jump to many points in the current file, including any identifier declared or defined in the file. You can also add items that aren't definitions or declarations.

- The class hierarchy menu is especially useful if you are new to Objective-C. You can quickly access the header files of the superclasses of any class in your application, which lets you investigate inherited methods and properties.

- The included files menu lists 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.

- The Counterpart button opens the counterpart of the current file or jumps to the symbolic counterpart of the currently selected 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.

  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.

Try it: Navigate through a file in the CFLocalServer project using the function pop-up menu. Then open its counterpart.

1. Double-click the `Common.c` file.

2. Place the cursor anywhere in the code.

3. Open the function pop-up menu and choose a function name.

4. Click the Counterpart button to open the header file associated with the `Common.c` file.

Two other text editor important features—code completion and Quick Help—are described next.

Code Completion

Code completion offers you an alternative to typing long method names, argument lists, and other symbols. When you've entered enough letters for Xcode to make a reasonable guess, you'll see the suggestion appear as dimmed text. To accept the suggestion, press Tab. To see a list of all possible completions, press the Esc key.

For symbols that include parameters, such as methods, Xcode optionally inserts placeholders for the arguments. To move to the next placeholder, press Control–/.

To disable or enable code completion, use the Automatically Suggest menu in the Code Sense pane of Xcode preferences.

Try it: Let Xcode complete your code.

1. Anywhere in the `Protocol.h` file, enter `NS` then press Esc to see the completion list that Xcode built so far.

2. In the list that appears, choose any item and press Return. Xcode enters the item you chose.

3. Delete the entered text so the program will compile.

Quick Help

Finding technical information about an unfamiliar technology or API symbol is an important and often time-consuming activity for software developers. To view the reference documentation for a symbol in your code, Option–double-click it. Quick Help opens inline to show you the reference documentation for that symbol. If you prefer to view the header file for the symbol, click the .h button at the top of the Quick Help window. If you would like to see the complete reference document, click the book icon.

Quick Help closes automatically when you start typing again. If you want to keep Quick Help open so you can use it like an inspector, simply move it.

Try it: Look up information about a symbol in the `Client.c` file.

1. In the `ConnectionGotData` function, find `kCFSocketDataCallBack`.

2. Insert the cursor in that symbol, then press the Option key and double-click the mouse.

The Documentation Window

You can use the Documentation window to search through all developer documentation (API reference, guides, sample code, technical notes, release notes, and technical Q&A documents). As you enter a term in the search field, Xcode simultaneously performs API reference, document title, and full text searches.

Try it: Find information using the Documentation window.

1. Open the Documentation window by choosing Help > Documentation.

2. In the search field, enter `NSMuta`.

3. Click a few different results to view the documentation.

4. Double-click any result to view it and also close the search results view.

Targets and Build Phases

A target contains the instructions for building a finished product from a set of files in a project—for example, a framework, library, application, or command-line tool. A simple Xcode project has just one target that produces one product. A larger development effort with multiple products may require a more complex project containing several related targets. For example, a project for a client-server software package may contain targets that create a client application, a server application, command-line tool versions of the client and server functionality, and a private framework that all the other targets use.

Build phases are instructions that Xcode follows to build a target. Each build phase consists of a list of input files and a task to be performed on each of those files. Common build phases include compiling files, linking object files, and copying resource files.

Try it: Look at the targets and build phases in the CFLocalServer project.

1. Click the disclosure triangle next to the Target group.

2. Click the disclosure triangles next to each of the targets in the Target group.

The CFLocalServer project has three targets. One builds a server program, one builds a client application, and another is a target whose purpose is to sequentially build the other targets. You can customize how a project is built by adding phases. To add a phase, choose Project > New Build Phases.

Build Settings

A build setting is a variable that contains information used to build a product. For each operation performed in the build process—such as compiling Objective-C source files—build settings control how that operation is performed. For example, the information in a build setting specifies the options Xcode passes to the tool—in this case, the compiler.

Each target can specify one or more sets of build settings, called build configurations. Targets are preconfigured with two build configurations, debug and release. The debug build configuration specifies build settings that generate products containing information that is useful during development, such as debug symbols. The release build configuration specifies build settings appropriate for products that are ready to deploy to your customers.

Try it: Find out which build configurations and settings are available for the CFLocalServer project.

1. In the Groups & Files list of the project window, select the project group and then click the Info button in the toolbar.

2. In the window that appears, click Build to display the Build pane of the Info window. You'll see the active configuration and a list of the build settings for that configuration.

3. Open the Configuration pop-up menu to see what other configurations are available. Choose a configuration you haven't yet looked at.

4. Compare the build settings for that configuration with the first configuration you looked at.

5. Click a build setting. Look at the bottom of the window to see its description.

Build Products

By default, Xcode places the built application bundle inside your project's build folder. You can run a built application from the Finder by double-clicking the application icon located in Debug or Release folder in your project's build folder. Which folder depends on whether the active build configuration is debug or release.

Try it: Check the build location for the the CFLocalServer project and then change it.

1. In the Groups & Files list of the project window, select the project group and then click the Info button in the toolbar.

2. Click General to see the current build location.

3. Change the location of the build products by selecting "Custom location", clicking Choose, and navigating to a location.

Preferences and Keyboard Shortcuts

Xcode preferences let you customize just about everything in the development environment including the behavior of the text editor, where build products are stored, code colors and indentation, code repositories, and the content shown in Quick Help.

One popular set of preferences is Key Bindings, which 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, and MPW.

> **Try it:** See what preferences are available.
>
> 1. Choose Xcode > Preferences.
>
> 2. Click each button at the top of the window to see the preferences that pertain to that topic—General, Code Sense, Building, and so on.

Xcode Workflow Tutorial

This short tutorial shows how to create an Xcode project for a Cocoa application called Hello that prints "Hello, World!" inside a window. Its primary purpose is to show you the workflow that you'll use for creating your own software with Xcode—from creating a project to getting an application to run. You'll also see how to fix compile-time errors and get an introduction to the debugger. Before reading this chapter, you should read "Xcode Features Overview" (page 9) so that you are familiar with Xcode terminology and how to use its major features.

The Hello, World Application: An Overview

Hello World is a simple application. When the user opens the application, a window appears that displays the text "Hello, World!" similar to what you see in Figure 2-1. Under the hood, the user interface consists of a window that contains a view object. View objects know how to display data. These objects have a built-in method that automatically draws to the view. You need to provide the code that draws "Hello World!"

Figure 2-1   The Hello World application
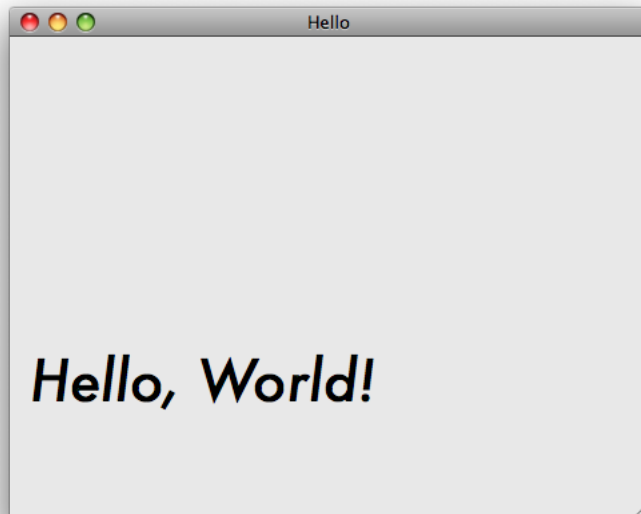
You'll use Cocoa, Apple's object-oriented language, to create a view object and implement the drawing routine. You don't need to know Cocoa to complete this tutorial, but you should be familiar with programming in some language, preferably a C-based or object-oriented language.

Creating an Xcode Project

Xcode includes a set of built-in project templates configured for building specific types of software products. A template project sets up the basic application environment by creating an application object, connecting to the window server, establishing the run loop, and so on.

To create an Xcode project for the Hello application using a template:

1. Open Xcode. (It's in the `/Developer/Applications` directory.)

2. In the Welcome to Xcode window, click "Create a new Xcode Project."

   If the Welcome to Xcode window does not appear, choose File > New Project.

   If you're curious, browse through the list of templates to see the variety of software products Xcode can build.

3. In the list on the left, select Application under Mac OS X.
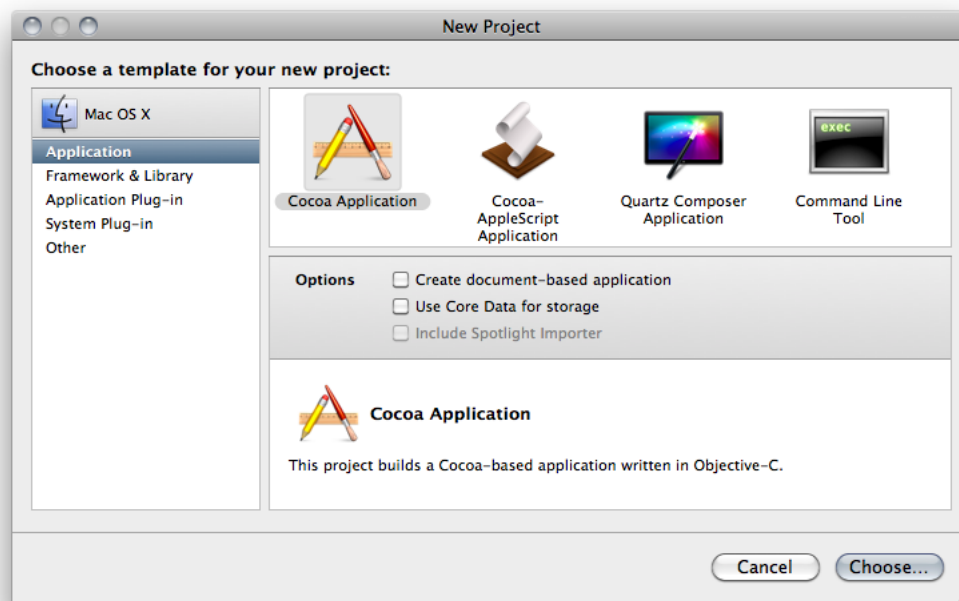
4. Select the Cocoa Application template and click the Choose button. (You don't need to select any of the options.)

5. Navigate to the location where you want the Xcode application to create the project folder.

6. Type `Hello` in the Save As field. Then click Save.

   The Xcode application creates the project and displays the project window described in the next section.

7. Open the `main.m` file and look at the `main` function.

Xcode automatically provides the `main` function. You don't need to modify it, but it's a good idea to understand its purpose.

This call performs most of the work of setting up the application and creates an instance of `NSApplication`:

```
return NSApplicationMain (argc, (const char **) argv);
```

The `NSApplicationMain` function also scans the application's `Info.plist` file, which is a dictionary that contains information about the application such as its name and icon.

Now it's time to get to the heart of creating the Hello World application—editing source files and setting up the user interface.

Editing Project Files

To modify the behavior of the application to print "Hello, World!" in its main window you need to:

- Create a view object (a subclass of `NSView`) to which you will later draw the "Hello, World!" greeting. You'll do this in Xcode.

- Add a view object user-interface element to the Hello application window. You'll do this in Interface Builder.

- Implement the drawing method provided by the `NSView` class. You'll do this in Xcode.

The following sections describe these tasks in detail.

Creating a Custom View Class

Cocoa always draws to objects known as views. The basic functionality of a view is implemented by the `NSView` class. This class defines the basic drawing, event handling, and printing architecture of an application. You typically don't interact with `NSView` directly. Instead, you create a custom subclass that inherits from `NSView` and then override the methods you need. The AppKit framework automatically invokes these methods.

Right now, all you need to do is create a subclass of `NSView`. Later, you'll write the code that performs drawing.

1. In the Groups & Files list of the Hello project window, select Classes.

2. Choose File > New File. The New File dialog appears.

   If you're curious, browse through the list of templates to see the variety of files Xcode can create for you.

3. Click Cocoa Class under Mac OS X, select the Objective-C class template, choose NSView from the Subclass menu, and click Next.
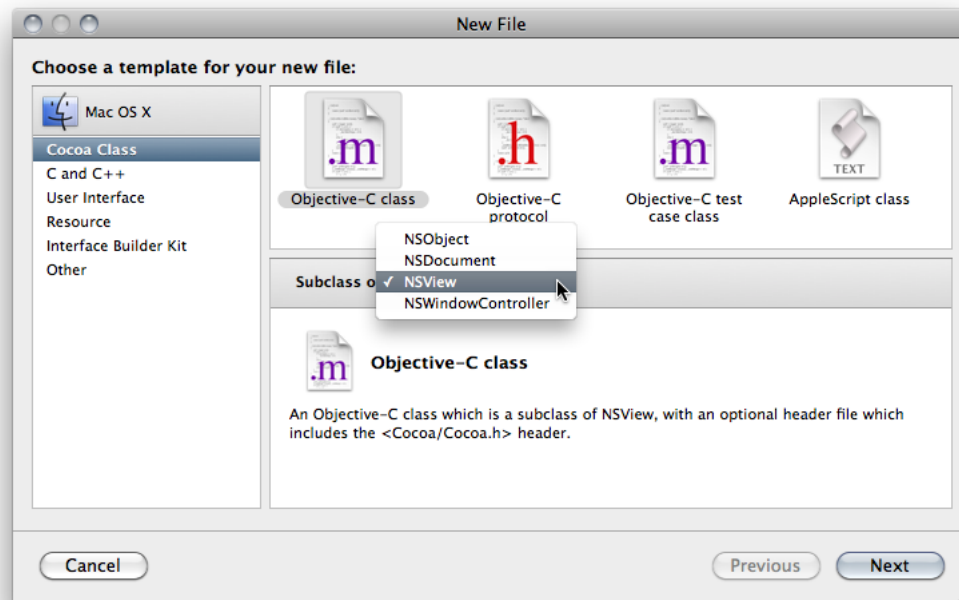
4. Enter `HelloView.m` in the File Name field. Make sure the option "Also create HelloView.h" is selected.

5. Click Finish.

    The Xcode application creates the source files and places them inside the Classes group in your project. You'll take a closer look at these later. Right now however, you'll turn your attention to the user interface.

Using Interface Builder

Interface Builder is Apple's graphical editor that you use to design user interfaces. It doesn't generate source code; instead it allows you to manipulate objects directly and then save those objects in an archive called a nib file. At runtime, when a nib file is loaded the objects are unarchived and restored to the state they were in when you saved the file.

You need to add an instance of the `HelloView` class that you just created in Xcode to the application window.

1. Open Interface Builder by double-clicking the `MainMenu.xib` file, which you can find in the detail view for the project group.

    Note: A nib file can have a `.xib` extension or a `.nib` extension. To find out more about the differences between these, see *Interface Builder User Guide*.

2. Double-click the Window icon located in the MainMenu.xib window.

When the user opens the Hello World application that you create, this is the window that appears.

3. Click inside the Window title bar and then choose Tools > Attributes Inspector.

The inspector lets you look at and modify characteristics of the user interface elements you add to your application. Notice that your window already has default settings that dictate its behavior.

Take a moment to click the buttons at the top of the inspector to see the other window characteristics—effects, size, bindings, connections, identity, and AppleScript. Later you'll change some of these other characteristics.
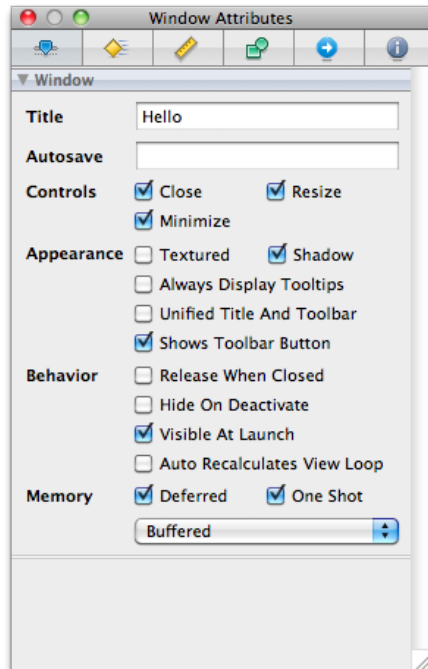
The window acts as a container for other user interface elements, like buttons, sliders, views, and text fields. For this application, you'll need to add a view to the window. It's the view that your application will draw to.

4. Choose Tools > Library to display the Interface Builder library.

The library contains prebuilt user interface objects and media that you can add to your application. Take a moment to look through the variety of objects that are available. As you select groups, you see icons for the objects that you can choose from. When you select an object, you see its description.

For the Hello World application, you need a custom view to draw to. You'll add that to the window next.

5. In the Library pop-up menu, choose Layout Views and then locate the Custom View icon.

6. Drag the Custom View element from the library to the Hello window.
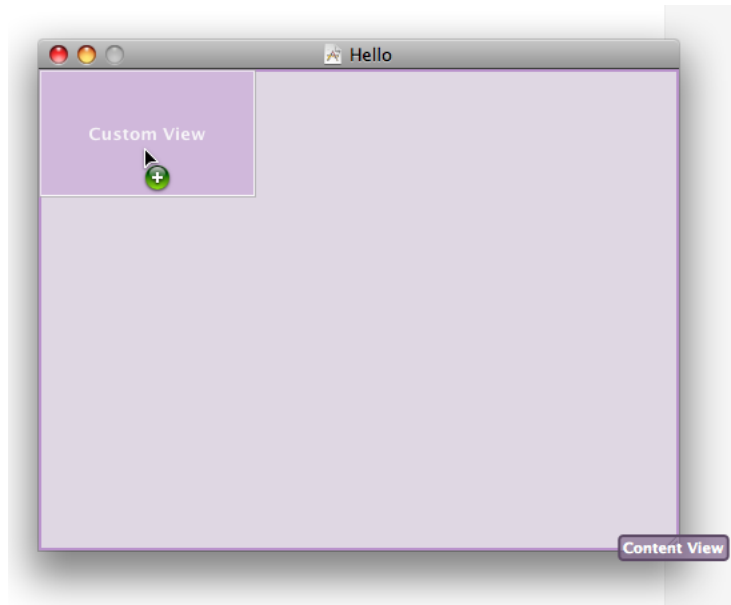
A custom view is a subclass of `NSView`. Later, in your code, you'll implement the drawing method (`drawRect:`) for the `HelloView` class that will draw to the custom view in the user interface.

7. Resize the Custom View element so that it occupies the entire content area of the Hello window.

8. Choose Tools > Identity Inspector. Then choose `HelloView` from the Class pop-up menu.

Recall that `HelloView` is a class that you created in Xcode. Interface Builder and Xcode communicate with each other behind the scenes. Classes you create in Xcode are known to Interface Builder, which is why you can choose it from the Class pop-up menu. Notice that the label for the view in the window changes from `Custom View` to `HelloView`.
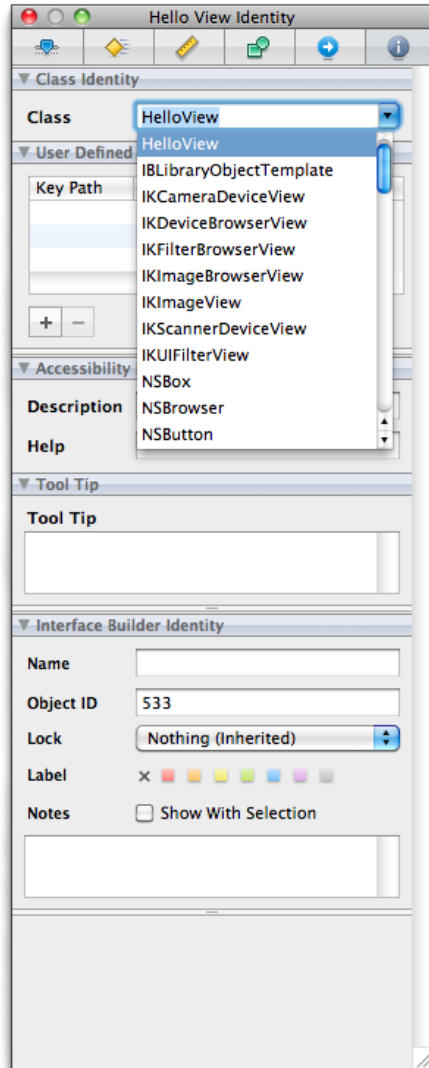
You resized the view so it is the same size as the window, but what happens if the user resizes the window? In some cases you might want the view to stay the same size. But in this case, you want the view to expand and contract with the window. You'll set that behavior next.

9. Select the view, and then choose Tools > Size Inspector.

The inspector provides information about the object that's currently selected. Now that you have two objects in your user interface—a window and a view, you want to make sure you have the appropriate one selected when using the inspector.

The Size inspector lets you enter precise values for positioning and sizing objects. You can also set the layout simply by moving and resizing the objects onscreen. Autosizing lets you specify how (and whether) objects change size as the enclosing window changes size. You'll set up autosizing next.
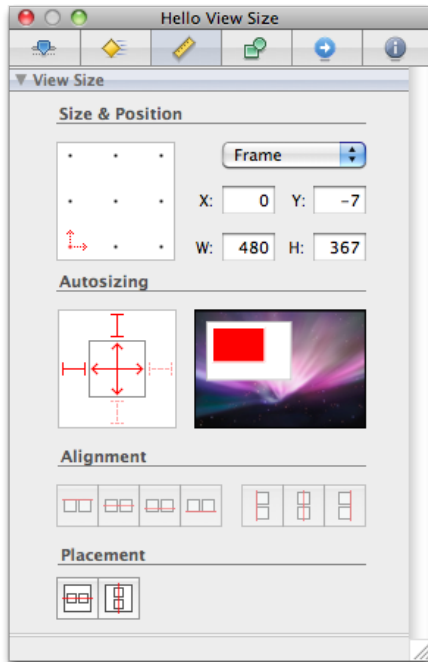
10. On the left side of the Autosizing area, click the vertical and horizontal lines in the inner square.

    Notice the dotted lines change to solid ones. Solid lines indicate the direction that the view should resize automatically. In this case, the view resizes automatically in both directions when the user changes the window size.

11. Save the Interface Builder file and quit Interface Builder.

There is much more to Interface Builder than you've seen here. When you write more advanced applications, you'll use the inspector to set bindings and connections to associate the code you write with the user interface objects that trigger that code.

Using the Text Editor

To edit the source code for the Hello project:

1. Open the Hello project window and select the Classes group.

   Your two custom source files—`HelloView.m` and `HelloView.h`—should be listed in the detail view.

2. Open `HelloView.m` in a separate window by double-clicking it in the detail view.

   The file should look something like Listing 2-1.

   Notice that Xcode automatically created the `initWithFrame:` and `drawRect:` methods for you.

Listing 2-1 Initial Implementation of the `HelloView` class

```
#import "HelloView.h"

@implementation HelloView

- (id)initWithFrame:(NSRect)frame {
    self = [super initWithFrame:frame];
    if (self) {
        // Initialization code here.
    }
    return self;
}

- (void)drawRect:(NSRect)rect {
    // Drawing code here.
}

@end
```

The method `initWithFrame:` is the designated initializer for the `NSView` class. It returns an initialized object. If your application needs to perform any other initializations related to the `HelloView` object, you'd add code to this method. Hello World is a simple application that doesn't require any custom initialization.

The `drawRect:` method is called automatically whenever the view needs to be drawn. The default implementation does nothing. You need to add whatever drawing code is appropriate for your application. You'll do that next.

> **Tip:** You can find out what a method does and get links to sample code, programming guides, and other useful information by using Quick Help. See "Quick Help" (page 13).

3. Insert these code lines as the body of the `drawRect:` method:

```
NSString* hello = @"Hello, World!";
NSPoint point = NSMake
```

The first line creates the string that you'll draw into the view.

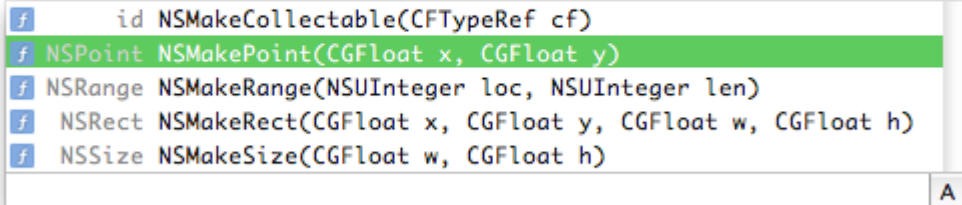The second line is incomplete, but next you'll see how to get help completing this line.

4. Position the cursor right after `NSMake` and press Esc. In the pop-up menu that appears, double-click `NSMakePoint` to choose it.

The menu contains the symbols Xcode knows about whose name start with `NSMake`.

```
- (void)drawRect:(NSRect)dirtyRect {
    NSString* hello = @"Hello, World!";
    NSPoint point = NSMake
}                    f        id NSMakeCollectable(CFTypeRef cf)
                     f  NSPoint NSMakePoint(CGFloat x, CGFloat y)
@end                 f  NSRange NSMakeRange(NSUInteger loc, NSUInteger len)
                     f   NSRect NSMakeRect(CGFloat x, CGFloat y, CGFloat w, CGFloat h)
                     f   NSSize NSMakeSize(CGFloat w, CGFloat h)
                                                                          A
```

5. Enter `15` for the `CGFloat x` parameter and `75` for the `CGFloat y` parameter. Then add a semicolon (`;`) to the end of the code line.

    This call creates a point at the coordinates values that you specify. This point will be the starting point for drawing the Hello, World text.

6. Complete the implementation of the `drawRect:` method so it looks like the one shown in Listing 2-2.

    This implementation draws the "Hello, World!" string starting at the view coordinates of (15, 75), using the Futura-Medium Italic font, size 42. The font name and size are added to a dictionary object, which is then passed to the `drawAtPoint:withAttributes:` method. This method does the actual drawing, using the attributes to added to the dictionary. After drawing the text, the `drawRect:` method releases the dictionary object to ensure proper memory management in Cocoa.

Listing 2-2   Implementation of the `drawRect:` method

```
- (void) drawRect:(NSRect) rect
{
    NSString* hello = @"Hello, World!";
    NSPoint point = NSMakePoint(15, 75);
    NSMutableDictionary* font_attributes = [NSMutableDictionary new];
    NSFont* font = [NSFont fontWithName:@"Futura-MediumItalic" size:42];
    [font_attributes setObject:font forKey:NSFontAttributeName];

    [hello drawAtPoint:point withAttributes:font_attributes];

    [font_attributes release];
}
```

7. Choose File > Save.

Building and Running the Application

The build system in Xcode handles the complex process of creating a finished product based on build settings and rules specified in each of the project's targets. When you initiate a build, Xcode begins a process that starts with the source files in your project directory and ends with a complete product. Along the way, Xcode performs various tasks such as compiling source files, linking object files, copying resource files, and so forth. After Xcode finishes building the product (and if it doesn't encounter any errors along the way), it displays "Build succeeded" in the status bar at the bottom of the project window.

To verify that the application runs:

1. Click the Build and Run button in the project window toolbar.

2. Verify that the application opens a window and displays "Hello,World!"

3. Press Command-Q to quit.

If you copied everything correctly, you won't have any compile-time errors.

Compile-Time Errors

Projects are rarely flawless from the start. By introducing a mistake into the source code for the Hello project, you can discover the error-checking features in Xcode.

To see how error checking works:

1. Open the `HelloView.m` file.

2. Remove the semicolon from the `point` definition code line, creating a syntax error.

3. Choose File > Save to save the modified file.

4. Choose Build > Build. As expected, this time the build fails.

5. The error and warning messages are displayed inline in the editor window.
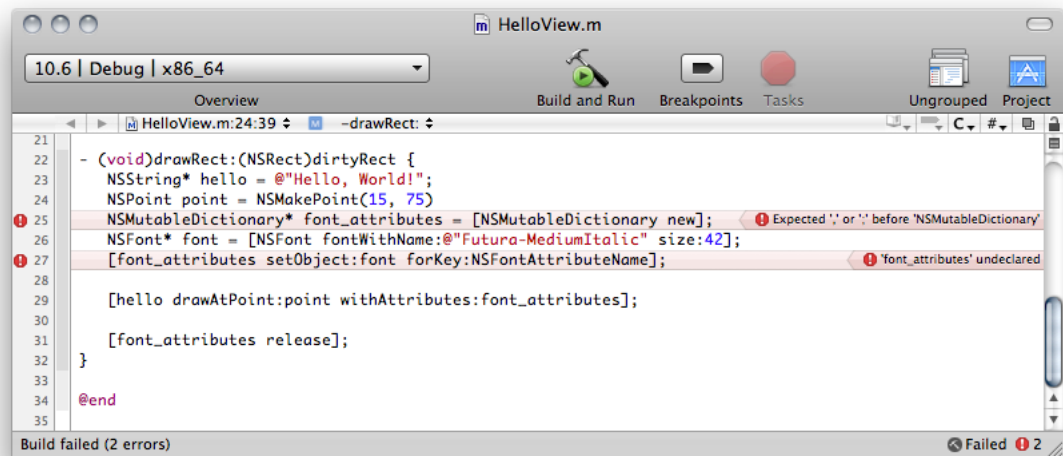
6. Fix the error in the source file, save, and build again. Notice that the error messages from the previous build are gone.

Runtime Debugging

Xcode provides a graphical user interface for GDB, the GNU source-level debugger. Debugging is an advanced programming topic that's beyond the scope of this tutorial, but it's useful to try out the debug command to see how it works. When you debug an application, you should make sure that Debug is the active build configuration.

To set a breakpoint and step through a block of code:

1. Open the `HelloView.m` file.

2. Find the line that defines the `hello` variable.

3. Set a breakpoint by clicking in the column to the left of the code line.
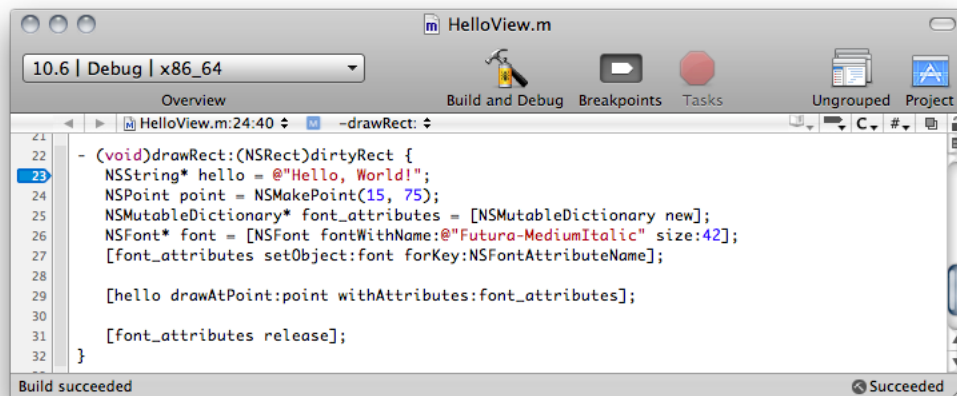
4. Choose Run > Debugger. Xcode opens the debugger.

5. Click Breakpoints and then click Build and Debug to have Xcode run the application in debug mode.

   Your application pauses at the breakpoint you set.
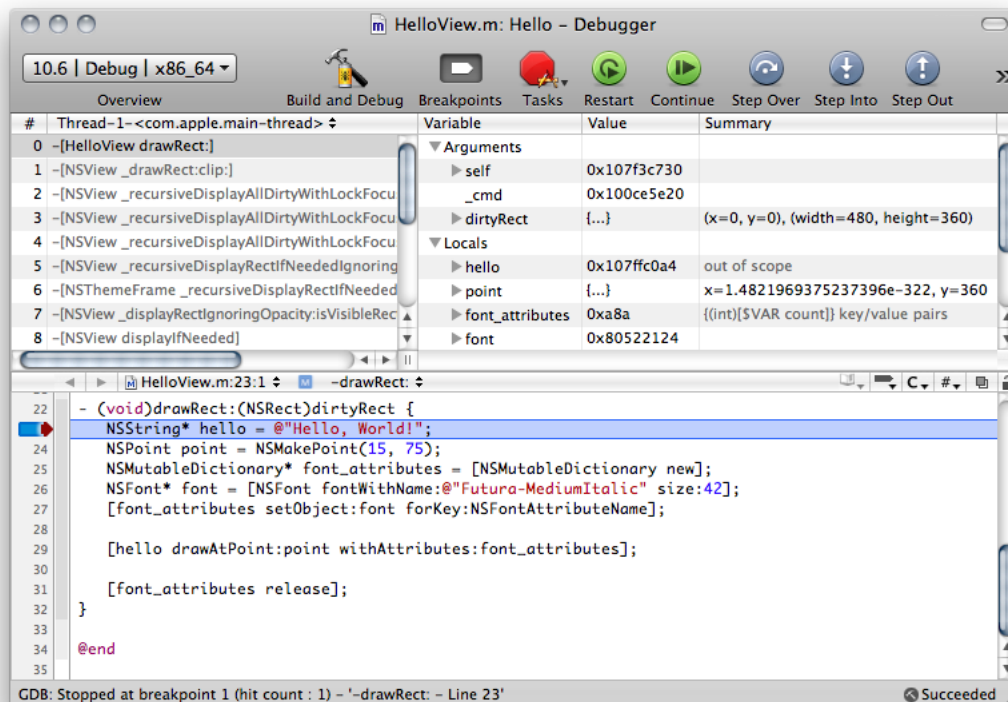
6. Using the Step Over button in the debugger toolbar, begin stepping through the code. As each line of code executes, you can examine the program's state. The value of a variable is sometimes drawn in red to indicate that the value was modified in the last step.

Notice that the debugger pauses before executing the indicated line. After each pause, you can add additional breakpoints or choose Debug > Restart to terminate the application and start a new debugging session.

Normally, Xcode just stops the execution of the program when it encounters a breakpoint. By specifying breakpoint actions, you make Xcode perform other actions, such as logging output to the console.

To add a breakpoint action to the breakpoint you added earlier:

1. Choose Run > Show > Breakpoints. Xcode displays the Breakpoints window.

2. In the detail view in the Breakpoints window, click the triangle to the left of the breakpoint you added earlier.

3. Click the add (+) button that appears below the breakpoint.

4. From the pop-up menu that appears in the line below the breakpoint, choose Sound.

5. From the second pop-up menu, choose the sound you want Xcode to play when it reaches that breakpoint.
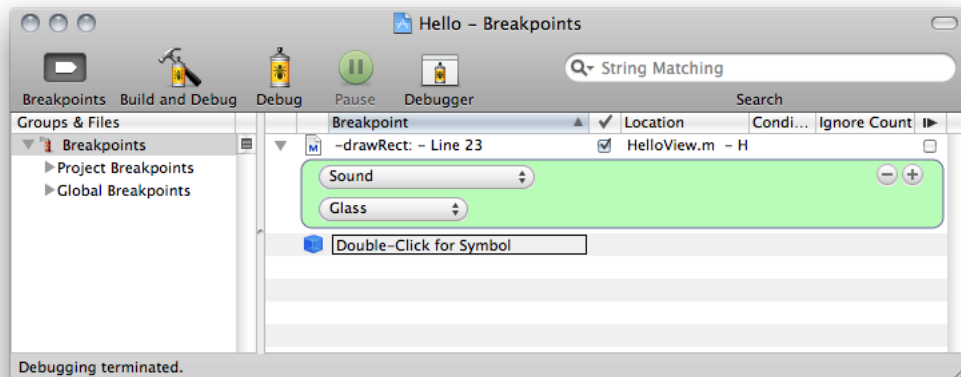
Now when execution reaches the breakpoint, Xcode plays a sound in addition to stopping the program.

For more information on breakpoints, see Managing Program Execution in *Xcode Debugging Guide*.

Recommended Reading

The ADC reference libraries contain API reference documentation, programming guides, tutorials, technical Q&As, sample code, and more. As a new Xcode user, the amount of information available can be overwhelming. This chapter points out some of the most important resources for each phase of development—from the design of your software product to its delivery.

Design

You may want to take a look at these documents before you create an Xcode project and start writing code:

- *Apple Human Interface Guidelines*. Use these guidelines to develop products that provide users with a consistent visual and behavioral experience across applications and the operating system. Make your application intuitive, friendly, and elegant.

- *Document-Based Applications Overview*. Read this overview if you want to create applications such as word processors, spreadsheets, image processors, and sound editors. Using a document-based application users can create identically contained, but uniquely composed, sets of data that can be stored in files.

- *Core Data Programming Guide*. Consider using the Core Data framework if your application stores and accesses data. Using it, you quickly define your application's data model in a graphic way and easily access the data model from your code. Core Data provides an infrastructure to deal with common functionality such as undo and redo and data persistence, allowing you to focus on adding innovative features to your application.

Code

When you are in the coding phase of your product, you might find these resources helpful:

- *Xcode Workspace Guide*. Read this document to find out how to customize your development environment to suit your work style. You can control the layout of the project window, the workings of the text editor, the content that appears in Quick Help, and many more aspects of the environment.

- *Xcode Unit Testing Guide*. Writing and performing unit tests as you write code can ensure that no errors or bugs are introduced as you make changes and improvements to your application's functionality. Find out how to use the unit-test environment of Xcode for testing Objective-C and C++ code.

- *Interface Builder User Guide*. Consult this document for in-depth information on using Apple's graphical tool to create your application's user interface. You'll see how to assemble user interface elements from a library of configurable objects and then connect those objects to your code.

- *SDK Compatibility Guide*. Do you need to develop software that can be deployed on, and take advantage of, features from specific versions of Mac OS X, including versions different from the one you are developing on? If so, consult this guide.

Build

When you are ready to have Xcode translate the source files and the instructions in a target into a product, the following guides and reference might be of value:

- *Xcode Project Management Guide*. Get an orientation to the Xcode build system by reading the Building Products chapter, which describes the most typical building tasks including how to:
  - ❏ View and change build settings
  - ❏ Build projects with single or multiple targets
  - ❏ Create a shared build directory
  - ❏ Set per-compiler flags

- *Xcode Build Setting Reference*. When you want to know more about a particular build setting, you can look up its definition in this document. You'll see how a target's build settings affect a build and how each setting relates to other settings.

- *Xcode Build System Guide*. Read this for a deep understanding of the Xcode build system and when you need to customize the build process. It describes how to use advanced features such as distributed builds and predictive compilation.

Debug

Finding and eliminating bugs in your code is a critical phase of the development process.

- *Xcode Debugging Guide*. Find out how to debug from each of the following:
  - ❏ The text editor
  - ❏ The mini debugger, which is a small window that's not too intrusive on the running application
  - ❏ The Debugger window, which is a specialized, traditional-style debugging environment
  - ❏ The GDB debugger console

Optimize Application Performance

Tools that gather performance data can help you to improve your application's performance.

- *Instruments User Guide*. Learn how to peer into your code as it's running and gather metrics about what it's doing. You can view and analyze the data Instruments collects in real time, or you can save that data and analyze it later. You can collect data about your application's use of the CPU, memory, file system, and the network, among other resources.

- *Shark User Guide*. Track down performance bottlenecks in your code. Use Shark to produce profiles of hardware and software performance events. You can analyze the profiles to get a better understanding of how your code works and interacts with the operating system.

Deliver

When your product is ready, it's time to package it for installation on a user's computer.

■ *Software Delivery Guide*. Find out how to deliver software through either manual or managed installations.

Finding and Viewing Documentation

Xcode provides several ways for you to look up information quickly:

- The Documentation window lets you browse and search items that are part of the ADC Reference Library as well as any third- party documentation you have installed.

- Quick Help is a lightweight window that provides the reference documentation for a single symbol without taking the focus away from the window you are working in. It pops up inline when you open it. When you start coding, it closes.

> **Tip:** You can turn Quick Help into a symbol inspector (which stays opens) by moving the window after it opens.

- The Help menu provides a search field that lets you search Xcode documentation and also provides commands that open the Documentation window and Quick Help.

To find ...	Do this ...
ADC reference libraries resources	Type a term in the search field of the Documentation window.
Quick Help for a symbol	Option–double-click the symbol in the text editor.
Xcode documentation	Type a term in the search field of the Help menu.
Header file for a symbol	Command–double-click the symbol in the text editor.
Reference document for a symbol	Command–Option–double-click the symbol in the text editor.

Document Revision History

This table describes the changes to *A Tour of Xcode*.

Date	Notes
2009-07-22	Made minor changes.
2009-05-12	Updated content for Mac OS X v10.6.
	Changed the title from *Xcode Quick Tour for Mac OS X*.
	Moved the detailed Interface Builder tutorial to *Interface Builder User Guide*.
	Moved the information in the "Fix and Continue" chapter to *Xcode Workspace Guide*.
	Added "Xcode Features Overview" (page 9), "Recommended Reading for Xcode Developers" (page 31), and "Finding and Viewing Documentation" (page 35).
2008-10-15	Added information about how to implement the Converter application.
2008-06-05	Made minor updates to several chapters. Changed the title from "Xcode Quick Tour Guide."
2006-11-07	Reorganized the "Finding Technical Information" chapter to emphasize API Search in the Xcode editor.
2006-07-24	Based Hello, World example on Cocoa instead of Carbon.
2006-01-10	Specified Xcode Tools version requirements.
2005-09-08	Updated introduction to clarify how to install Xcode Tools.
2005-06-06	Updated for Mac OS X v10.4. Changed title from "A Quick Tour of Xcode."
2003-08-28	First public version of this document.
2003-06-20	Released as a preliminary document for WWDC 2003.

Index