Xcode Quick Tour for Mac OS X

Tools > Xcode

2008-10-15

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Simultaneously published in the United States and Canada.

Contents

Figures and Listings

Introduction to Xcode Quick Tour for Mac OS X

Note: This document was previously titled *Xcode Quick Tour Guide*.

Xcode Tools is the developer tools package for Mac OS X. This package includes an integrated suite of software development tools, including 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 in Mac OS X. (Elsewhere in this document, the name Xcode usually refers to the Xcode application.)

If you're new to Xcode and you want to develop applications for Mac OS X, reading this document is a good way to get started. This document gives you a hands-on introduction, in the form of four short tutorials, to the Xcode application and some of its companion tools. To get the most out of these tutorials, you should already be familiar with C programming and the Mac OS X user interface. You don't need any previous experience with Mac OS X software development.

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

Important: This document is targeted for Mac OS X v10.5 or later, and Xcode 3.1 or later. Before continuing, make sure your development environment meets these requirements. To find out which version of Xcode is installed on your computer, launch the Xcode application (`/Developer/Applications`) and choose Xcode > About Xcode.

Organization of This Document

This document contains the following chapters:

- "Creating a Project" (page 9) introduces you to some of the basic features in Xcode.
- "Finding Technical Information" (page 29) shows you how to quickly find and display technical information in Apple's developer documentation library.
- "Designing a User Interface" (page 39) demonstrates how to use Interface Builder to design a user interface for a Cocoa application.
- "Using Fix and Continue" (page 55) describes how to use Xcode's patching facility to modify a running application during a debugging session.

See Also

Here are some additional resources you may want to read:

- For information about all aspects of software development in Mac OS X, visit Apple Developer Connection.
- If you're interested in developing applications using Cocoa, see *Cocoa Application Tutorial* and *Cocoa Fundamentals Guide*.
- For an overview of the software development tools in Xcode, see *Xcode Overview*.
- To learn more about the Xcode application, see *Xcode Workspace Guide*.
- If you want an introduction to Mac OS X system architecture and technologies, see *Mac OS X Technology Overview*.
- The principles and conventions of Mac OS X user interface design are covered in *Apple Human Interface Guidelines*.

Creating a Project

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.

Xcode is a great application for creating and managing projects. Xcode can be used to build a wide variety of software products, ranging from Carbon and Cocoa applications to kernel extensions, libraries, and Mac OS X frameworks.

This short tutorial shows how to create an Xcode project for a Cocoa application called Hello that prints "Hello, World!" inside a window. Along the way, you get a chance to explore some of the basic features of Xcode.

To help you troubleshoot problems as you follow the tutorial, this document includes the finalized Hello project in a companion archive named XcodeQuickTour_companion.zip.

Creating an Xcode Project

Xcode includes a set of built-in project templates configured for building specific types of software products. When creating a project, you can save time by starting with the appropriate template.

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

1. Launch Xcode.

   To launch Xcode, find it in the `/Developer/Applications` directory and double-click its icon (see Figure 1-1). If the Welcome to Xcode window opens, close it for now.

   Figure 1-1 The Xcode application icon

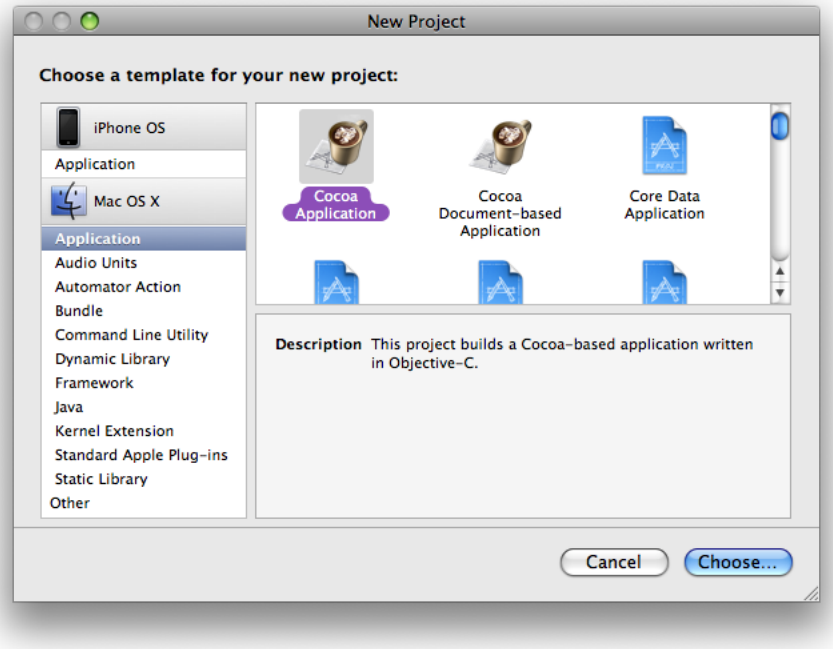
2. Choose File > New Project. The New Project dialog appears.

   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 Mac OS X Application (as shown in Figure 1-2.)

4. Select the Cocoa Application template and click the Choose button.

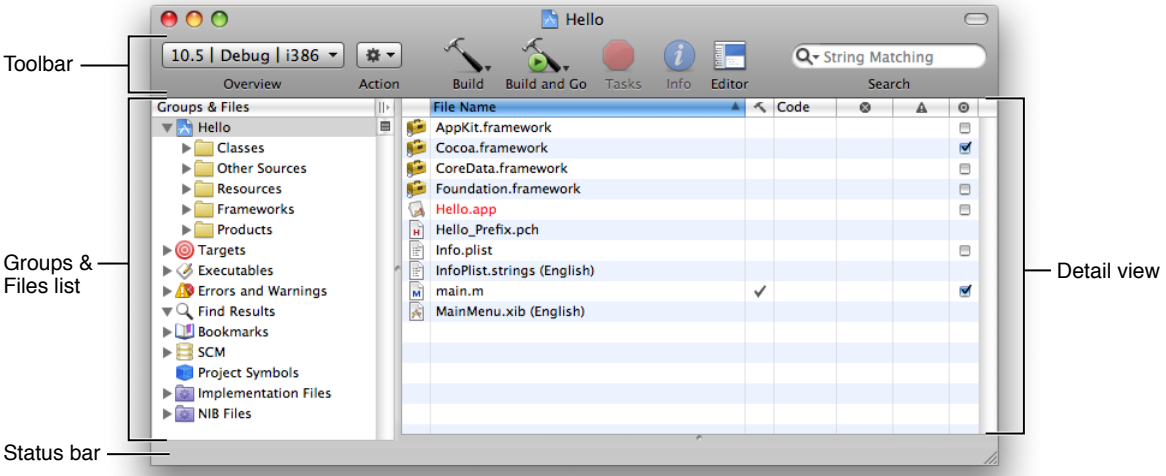
Figure 1-2 The New Project dialog



5. Navigate to the location where you want the Xcode application to create the project folder.
6. Type `Hello` in the Save As field.
7. Click Save.

   The Xcode application creates the project and displays the project window, as shown in Figure 1-3.

Figure 1-3 Hello project window



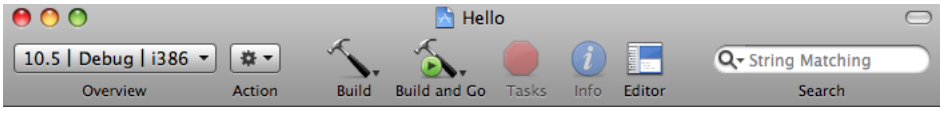
The Project Window

The project window is the control center for an Xcode project. This section briefly describes the components of the project window.

The Toolbar and Status Bar

The project window toolbar contains buttons and other controls you can use to perform common operations. Figure 1-4 shows the default toolbar.

Figure 1-4 The project window toolbar



- The Overview drop-down menu selects target settings such as the active SDK, configuration, and architecture.
- The Action drop-down menu displays actions you can perform on the currently selected item. You get the same menu when you Control-click an item.
- Build buttons initiate actions such as building, cleaning, running, and debugging.

> **Note:** A toolbar button with a drop-down triangle, such as Build or Build and Go, has additional commands associated with it. Pressing one of these buttons allows you to choose a different command from a drop-down menu.

- The Tasks button allows you to stop any operation in progress.
- The Info button opens an Info window for inspecting and editing groups, files, and targets in your project.
- The Editor button shows or hides the text editor in the project window.
- The Search text field filters the items currently displayed in the detail view.

Located at the bottom of the project window is the status bar, which displays status messages for the project, such as whether a build is successful.

Groups & Files

The Xcode application uses various groups to organize the files and information in a project. These groups are visible in the Groups & Files list, shown in Figure 1-3 (page 11). To see the contents of a group, select the group or click its disclosure triangle.

Groups are flexible—they don't need to 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.

Groups with plain icons are static groups. The items in a static group stay put until you move them. Groups with gears or other fancy icons are dynamic groups, called **smart groups**, that show items with a particular characteristic. The items in a smart group can change as you perform various actions in your project.

Here are some of the basic groups:

- The project group organizes all the components needed to build your product. This group is always listed first, and its name is the same as the project.
- Inside the project group are subgroups that contain your project's source files, resource files, frameworks, and products.
- The Targets group contains all the build targets in your project. A build target is a blueprint for building a product from a set of specified files and libraries.
- Errors and Warnings is a smart group that contains any error and warning messages generated during the most recent build.

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.

Editing Project Files

This section shows how to modify the behavior of the application to print "Hello, World!" in its main window.

To implement this new behavior, you:

- Create the `HelloView` class as a subclass of `NSView`
- Add a `HelloView` user interface element to the Hello application window

- Use the `drawRect:` method of the `HelloView` class to draw the textual greeting

The following sections describe these tasks in detail.

Creating a Custom View Class

In Cocoa, all drawing is done in objects known as **views.** The basic functionality of a view is implemented by the `NSView` class. Subclasses of `NSView`, such as `NSButton` and `NSTextView`, add functionality tailored for specific user interface objects.

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. Select the Cocoa Objective-C `NSView` subclass template, as shown in Figure 1-5, and click Next.

Figure 1-5 The New File dialog

4. Enter `HelloView.m` in the File Name field. Make sure the option "Also create HelloView.h" is checked.
5. Click Finish. The Xcode application creates the source files and places them inside the Classes group in your project.

Using Interface Builder

Interface Builder is Apple's graphical editor for designing user interfaces. Interface Builder makes it easy to design an interface that adheres to the Aqua layout guidelines and takes advantage of the newest Cocoa technologies.

To familiarize yourself with the task of creating user interfaces with Interface Builder, you're going to add an instance of your `HelloView` class to the application window.

1. Open Interface Builder.

    a. In the Groups & Files list in Xcode, select the project group and locate the file `MainMenu.xib` in the detail view. For historical reasons, an Interface Builder document such as `MainMenu.xib` is called a **nib file**.

    b. Double-click this nib file to launch Interface Builder and open the Hello interface.

2. Find the main application window and change its title.

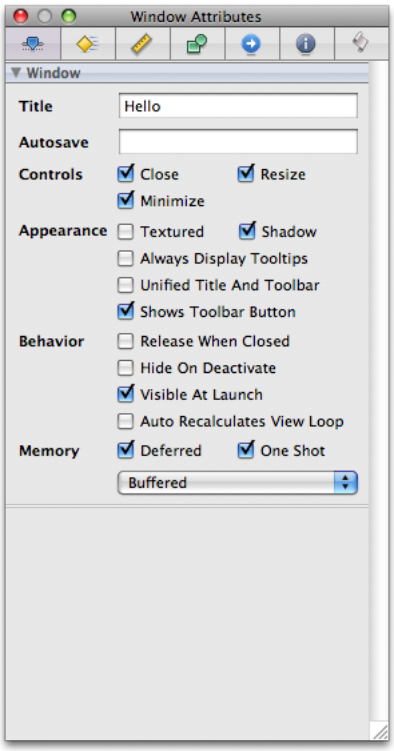
    Interface Builder displays several windows. The main window is the one with "Window" in the title bar. Change the title of the main window as follows:

    a. Click inside the Window title bar.

    b. Choose Tools > Attributes Inspector. Figure 1-6 shows the window attributes pane in the inspector window.

c. Enter `Hello` in the Title field and press Return.

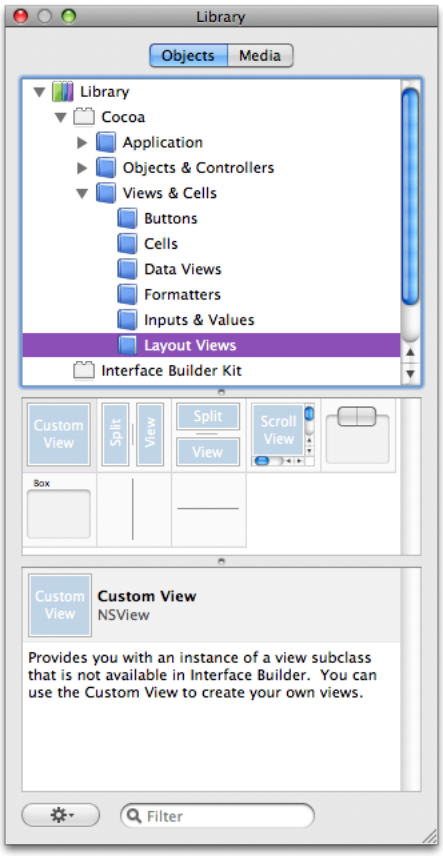
Figure 1-6 The Interface Builder attributes inspector



3. Add a `HelloView` element to the Hello window.

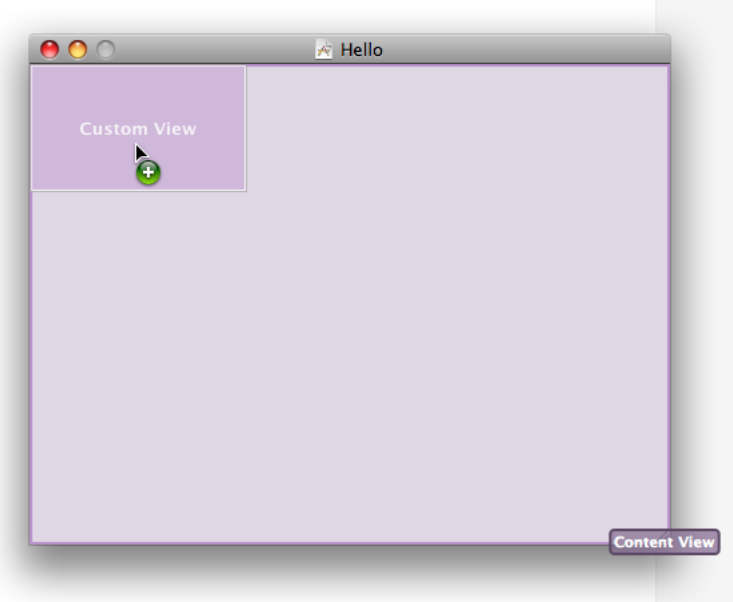
a. In Interface Builder, Choose Tools > Library to display the library palette. Select the Objects tab, and disclose Cocoa > Views & Cells. Then select Layout Views, as illustrated in Figure 1-7.

Figure 1-7 Interface Builder library palette



b. From the Layout Views pane, drag the Custom View element to the Hello window, as shown in Figure 1-8.

Figure 1-8 Adding a user interface element to a window in Interface Builder

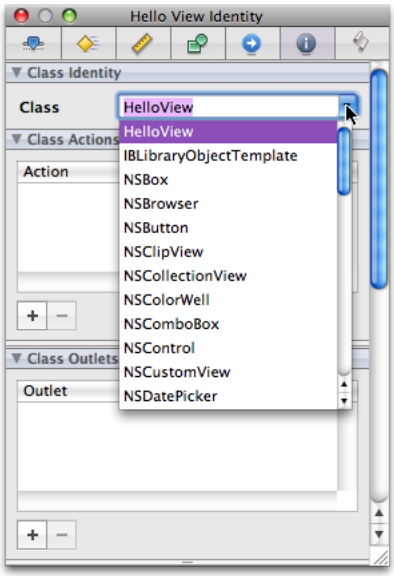


c. Resize the `CustomView` element so that it occupies the entire content area of the Hello window.

d. Choose Tools > Identity Inspector.

e. Use the Class pop-up menu to select `HelloView` in the Class list, as shown in Figure 1-9.

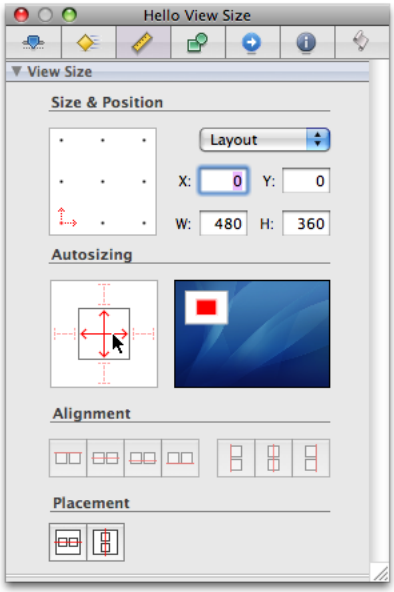
Figure 1-9 Selecting the class of a user interface element in Interface Builder



f. Choose Tools > Size Inspector.

g. In the Autosizing area, click the vertical and horizontal lines in the inner square, as shown in Figure 1-10.

Figure 1-10 Specifying the autosizing behavior of a user interface element in Interface Builder



Save the nib file and quit Interface Builder.

Using the Text Editor

Xcode has a built-in text editor that supports multiple buffers and windows. For convenience, a source file can be edited in a separate editor window or directly inside the project window.

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 should be listed in the detail view.

2. Open `HelloView.m` in an editor by double-clicking it in the detail view.

   The file should look something like Listing 1-1.

Listing 1-1 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
```

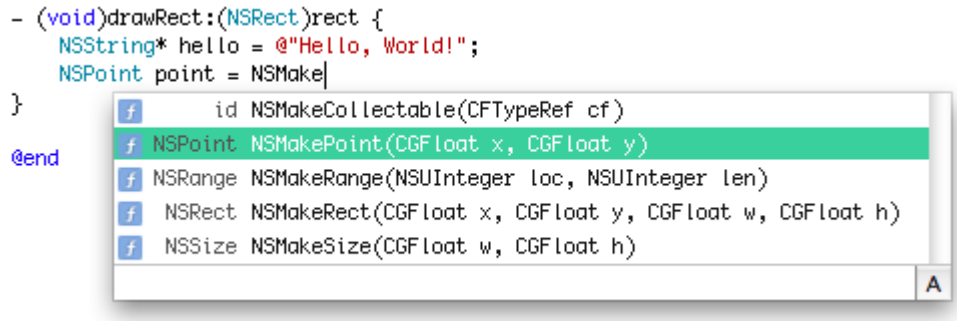
3. Insert these code lines as the body of the `drawRect:` method:

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

4. Position the cursor right after `NSMake` and press Escape. A pop-up menu appears.

   The menu contains the symbols Xcode knows about whose name start with `NSMake`, as shown in Figure 1-11.

Figure 1-11 The code-completion pop-up menu



   This is an example of **code completion**. When Xcode underlines the text you've typed, it has compiled a list of symbols whose name starts with the letters you've typed so far. You can continue typing or press Escape to view this list.

> **Note:** When a code-completion list contains symbols that have parameters or return values, you can view this information by choosing Xcode > Preferences > Code Sense and selecting the Code Completion checkboxes.

5. Complete the `point` assignment expression:

   a. Choose `NSPoint NSMakePoint(CGFloat x, CGFloat y)` from the code-completion pop-up menu.

   b. Replace `CGFloat x` with `15`.

   c. Replace `CGFloat y` with `75`.

d. Add a semicolon (;) to the end of the code line.

6. Finish entering the implementation of `drawRect:`, shown in Listing 1-2.

Listing 1-2 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. Save your changes by choosing File > Save.

Building the Application

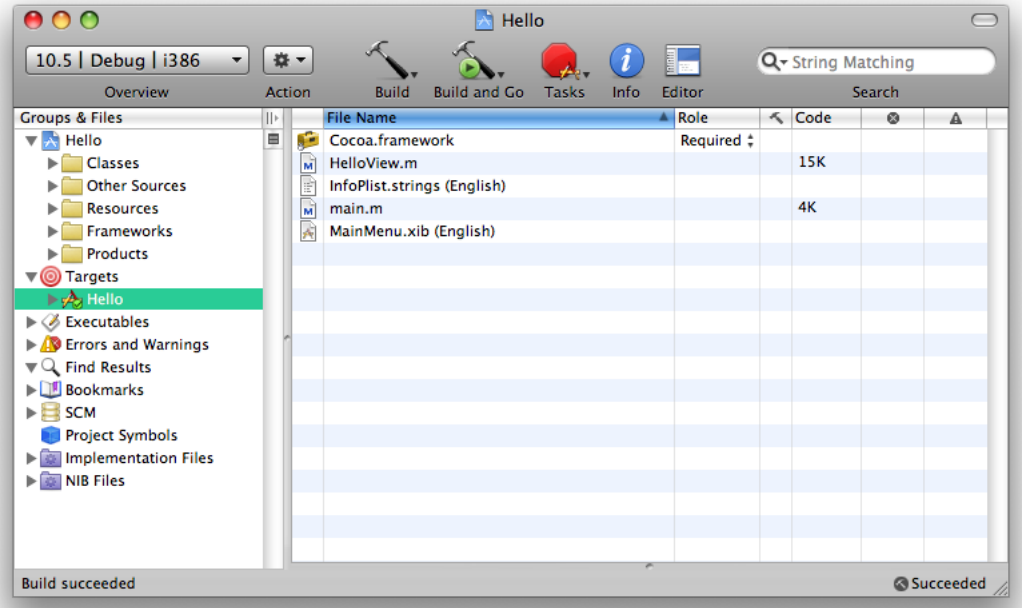
When you initiate a build, Xcode begins a process that starts with the source files in your project directory and ends with a complete product. Along the way, Xcode performs various tasks such as compiling source files, linking object files, copying resource files, and so forth.

To build the Hello application:

1. Choose Project > Set Active Build Configuration > Debug.
2. From the Build menu, choose Build, or click the Build button in the project window's toolbar.

After Xcode finishes building the product (and if it doesn't encounter any errors along the way), it displays "Build succeeded" in the project window's status bar, as shown in Figure 1-12. See "Compile-Time Errors" (page 23) for details on handling build errors.

Figure 1-12 A message in the project window's status bar



One of the build tasks is to copy nib files, sounds, images, and other resources from the project to the appropriate places inside the application bundle. An **application bundle** is a directory that contains the application executable and the resources needed by that executable. This directory appears in the Finder as a single file that can be double-clicked to launch the application.

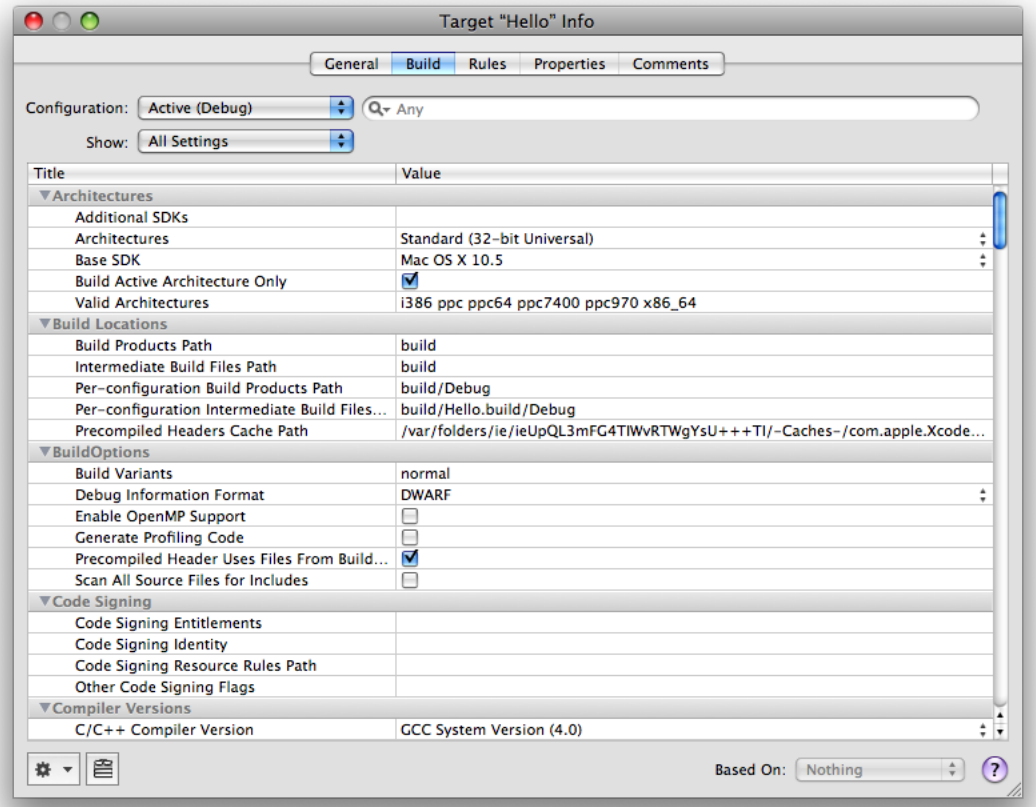
The build system in Xcode handles the complex process of creating a finished product based on build settings and rules specified in each of the project's targets. A target represents a single product, and Xcode supports multiple targets in a project.

Each target can specify one or more sets of build setting specifications, 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 for regular use.

To see your application's current build configuration:

1. In the Groups & Files pane of the project window, select the Hello target and choose File > Get Info.
2. In the window that appears, select the Build tab, as shown in Figure 1-13. This figure shows some of the build settings defined in the Debug build configuration of the Hello target.

Figure 1-13 The Debug build configuration in the Hello target info window



For more information on build configurations, see *Xcode Build System Guide*.

Running the Application

Now you have an application that's ready to run. Xcode places the application bundle in a location specified in your project's settings—in this case, inside your project folder. (See, within your project's build folder, a folder named Release.)

To verify that the application runs:

1. Choose Run > Go (Run), or click the Build and Go button in the project window's toolbar.

2. Verify that the application opens a window, displays “Hello,World!” and waits for user interaction.

Figure 1-14 Main window for the Hello application

3. Close the window by clicking the window’s close button or typing Command-W.
4. Type Command-Q to quit.

You may also run the application from the Finder by opening the Release folder in the project’s build directory, and double-clicking Hello.

Compile-Time Errors

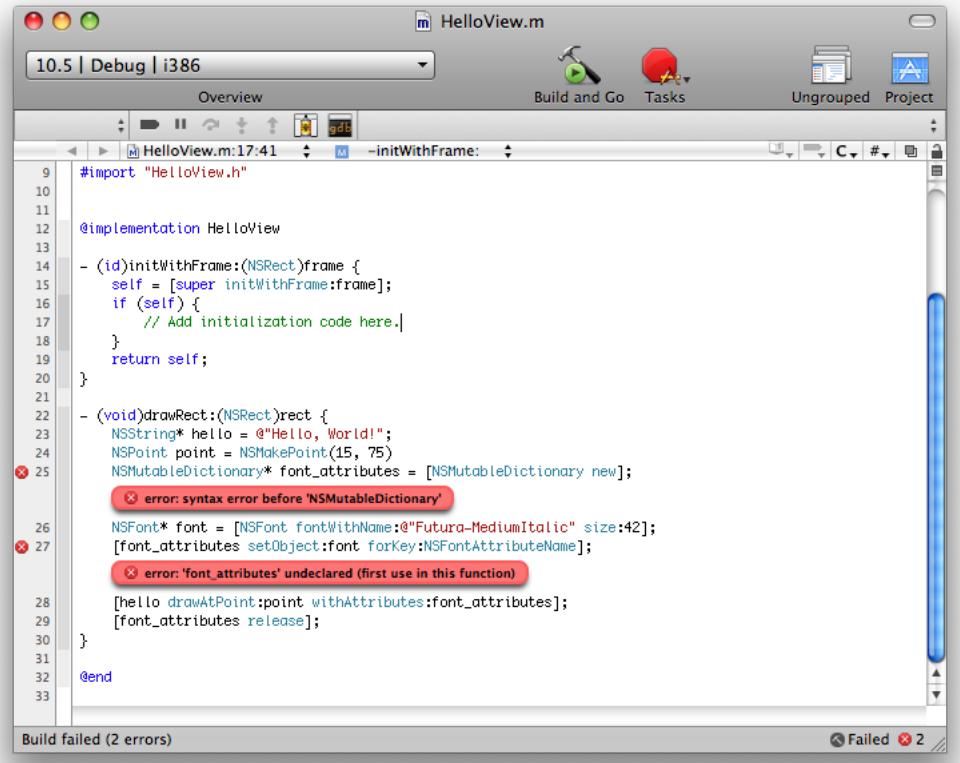
Projects are rarely flawless from the start. By introducing a mistake into the source code for the Hello project, you can discover the error-checking features in Xcode.

To see how error checking works:

1. Open `HelloView.m` in either the project window or a separate editor window.
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, as shown in Figure 1-15.

Figure 1-15 Error and warning messages



6. Fix the error in the source file, save, and build again. Notice that the error messages from the previous build have been cleared.

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. Before you can debug the Hello application, you need to set Debug as the active build configuration. Choose Project > Set Active Build Configuration > Debug.

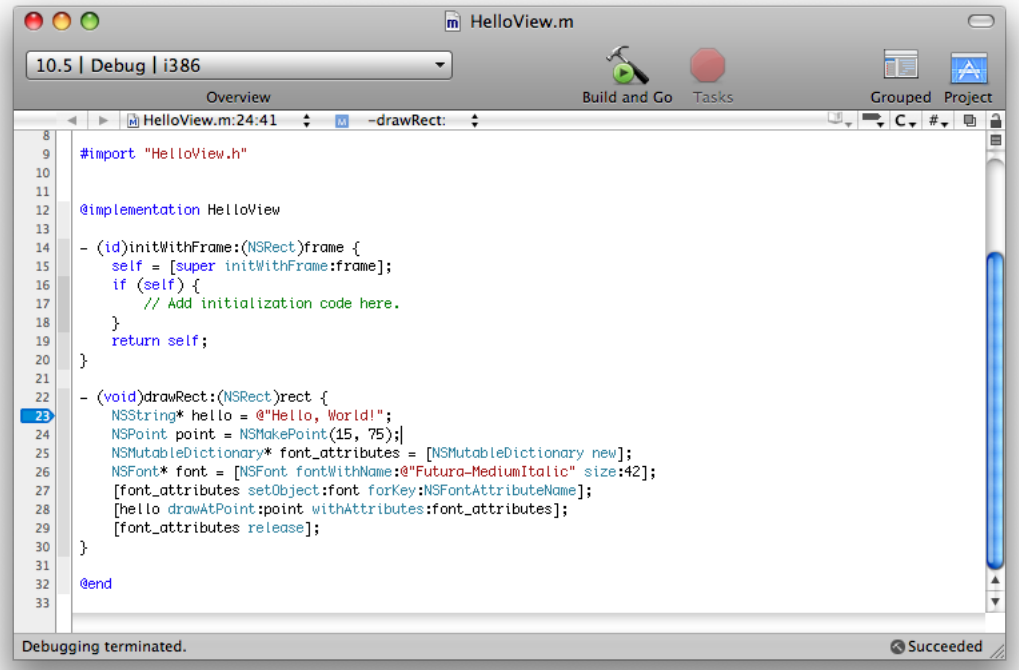
To set a breakpoint and step through a block of code:

1. Open the `HelloView.m` file. As before, you can open the file inside the project window or in a separate editor window.

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

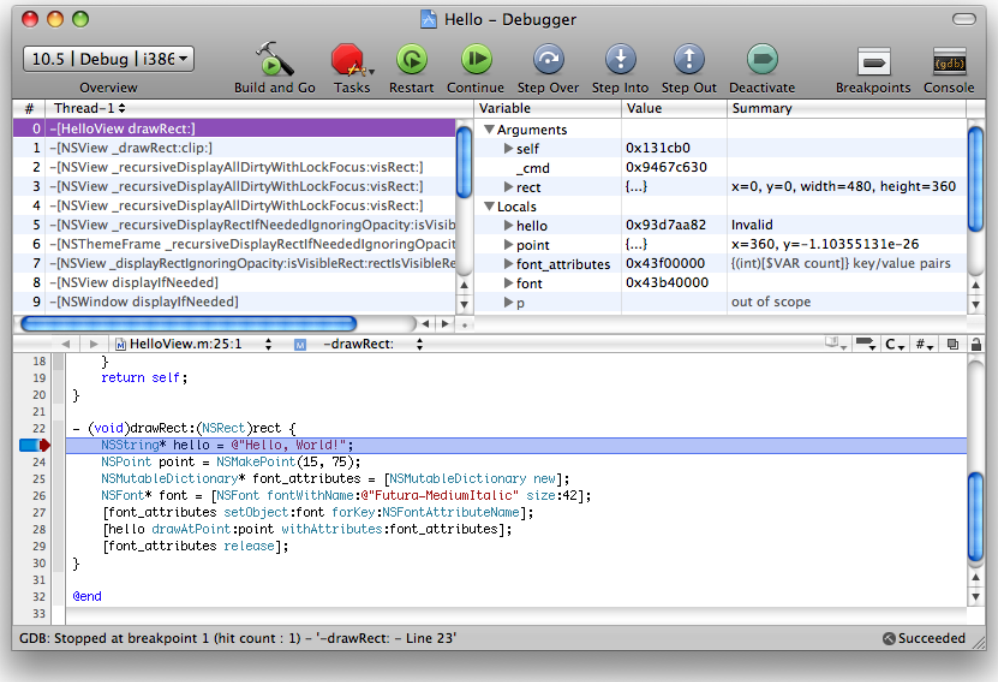
3. Set a breakpoint by clicking in the column to the left of the code line, as shown in Figure 1-16.

Figure 1-16 Setting a breakpoint



4. Choose Run > Debugger. Xcode opens the debugger window. Click the Build and Go button, and Xcode runs the application in debug mode, pausing at the breakpoint you set (see Figure 1-17).

Figure 1-17 The Debug window



5. Using the Step Over button in the Debugger window's 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 use the Debug > Restart command to terminate the application and start a new debug session.

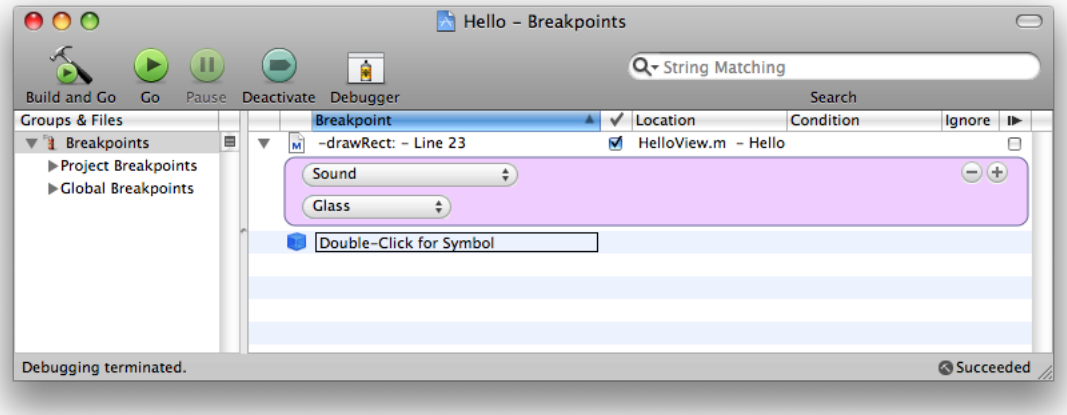
Normally, Xcode just stops the execution of the program when it encounters a breakpoint. By specifying **breakpoint actions**, you make Xcode perform other actions, such as logging output to the console.

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 first pop-up menu that appears below the breakpoint, choose Sound.
5. From the second pop-up menu, choose your preferred breakpoint sound.

Figure 1-18 shows a breakpoint action that plays the Glass sound when the breakpoint is reached.

Figure 1-18 A breakpoint action



Now, Xcode plays a sound—in addition to stopping the program—when execution reaches the breakpoint.

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

Summary

This tutorial provided a brief introduction to the Xcode application. It showed how to use Xcode to manage projects, build products, enter source code using code completion, find and correct build errors, and debug an application using breakpoints and breakpoint actions.

Finding Technical Information

Finding technical information about an unfamiliar technology or API symbol is an important and often time-consuming activity for software developers. Xcode includes an array of features to help you quickly find technical information as you work on your project.

The Xcode documentation window provides a consistent, intuitive user interface and a wide range of search options for viewing the HTML-based documentation in the ADC Reference Library. Xcode provides several ways for getting you to the information you seek: focused API search, title search, and text-based search.

- **API Search** is the fastest way to get to reference documentation from an editor window or in the Documentation window. "Searching For API Documentation" (page 31) describes this method.
- **Title Search** finds documents with titles that start with or contain your search string.
- **Full-Text Search** provides a way to find conceptual and reference documentation on a particular subject. With this method you can look through the set of documents that deal with the area in which you're interested. "Searching for Relevant Documentation" (page 33) delves into this search method.

This chapter shows you how to use Xcode to find technical information. It also explains how to view the header files in Mac OS X frameworks and how to update the ADC Reference Library on your computer.

Downloading Documentation

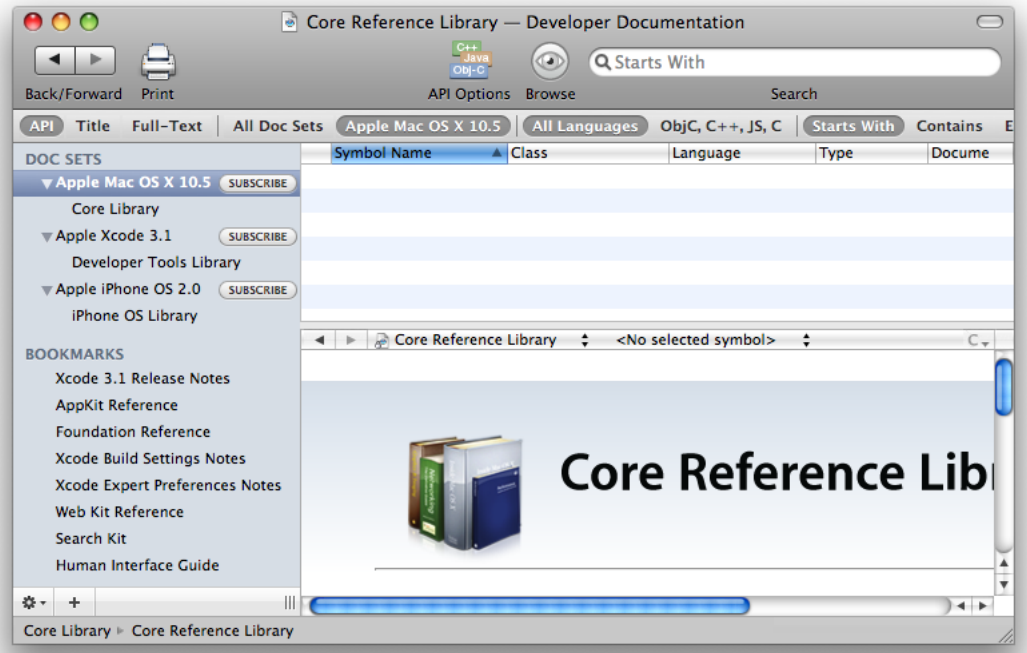
Xcode hosts a library of documents, code examples, technical notes, and release notes about development for Mac OS X using Apple technologies. To view this documentation library, open the Xcode documentation window:

1. Launch the Xcode application.
2. Choose Help > Documentation.

Note: You can also view the latest version of this library online at http://developer.apple.com/mac/. When you click this link, you may be taken to the Apple Developer Connection (ADC) sign-in webpage in your web browser. If you're an ADC member, sign in. Otherwise, sign up as an ADC member. ADC membership is available for free.

The first time you open the Xcode documentation window, Xcode displays a Subscribe button for each documentation feed in the pane on the left, as shown in Figure 2-1.

Figure 2-1 Subscribing to documentation feeds

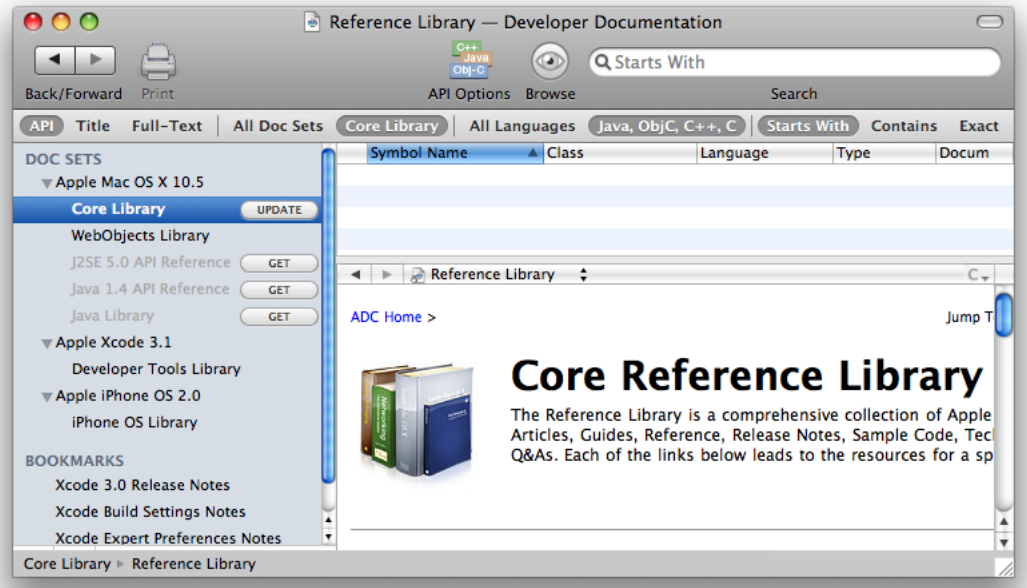


To subscribe to the Mac OS X documentation feed:

1. Click the Subscribe button next to Apple Mac OS X 10.5.
2. In the authentication dialog, sign-in with your ADC member name and password.
3. After Xcode downloads the core documentation for Mac OS X, note that other Mac OS X documentation sets are listed along with Get buttons. If you want any of these additional documentation sets, click the corresponding Get Button.

The documentation library is updated more often than new versions of Xcode are released. To keep your local copy of the library up to date, Xcode periodically checks for updates to the library's content. When an update is available, Xcode displays an update button in the documentation window, as shown in Figure 2-2. Before you click a documentation update button, make sure you're able to authenticate as an administrator of your computer.

Figure 2-2 Updating documentation sets



You can also check for updates at any time in the action menu at the bottom of the documentation window. In that menu, choose Documentation Set Updates > Check Now. See *Xcode Workspace Guide* for more information about downloading documentation.

Searching For API Documentation

Xcode makes it easy to find the API documentation for any Apple-defined symbol—including functions, classes, methods, data types, and constants.

Searching from the Documentation Window

The search field in the documentation window allows you to display a list of all known symbols that match a particular search string. As you type in the search string, Xcode dynamically builds and displays the list.

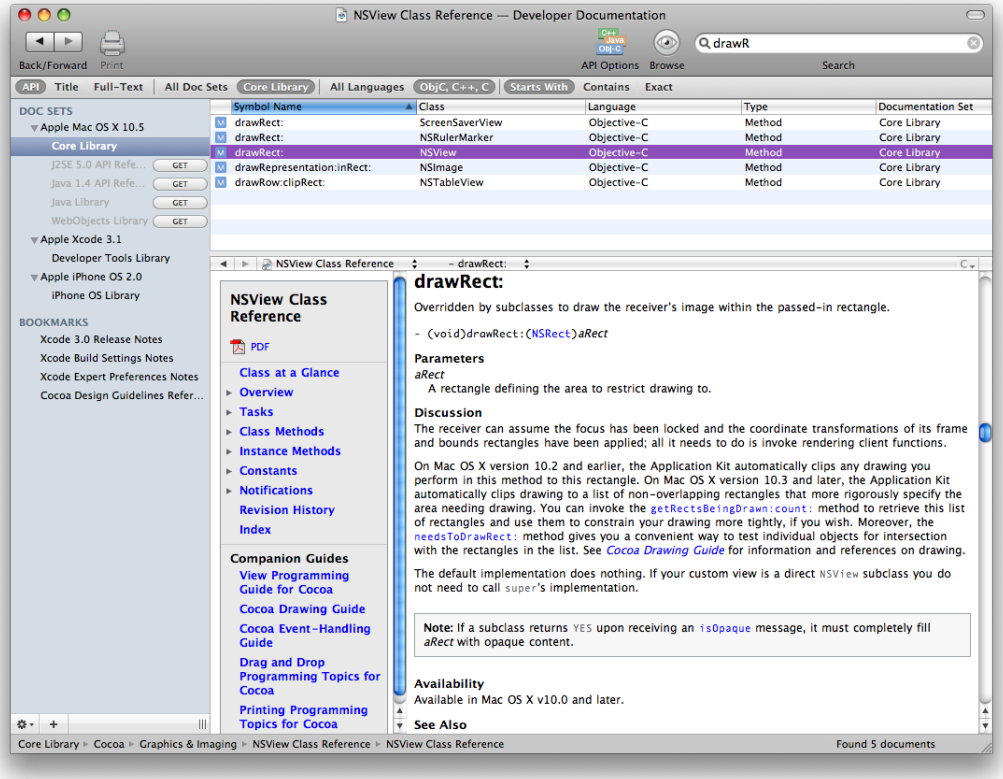
To find API documentation using a search string:

1. Launch Xcode and choose Help > Documentation.
2. In the DOC SETS list, select Core Library to define the scope of the search.
3. In the search bar, click the API and Starts With buttons.
4. Click in the search field and begin typing the name of the `drawRect:` method. Notice that each time you type a character, the list of matching symbols is reduced in size.

5. Stop typing when the list of matching symbols is reduced to just a few entries.

6. Select the entry for `drawRect:` in the `NSView` class. As shown in Figure 2-3, Xcode finds and displays the API documentation for this method in the content pane.

Figure 2-3 Using the API documentation search option in Xcode



Searching from a Text Editor Window

As a convenience, Xcode can also locate the API documentation for a symbol directly from a text editor window.

1. Open the Hello project window.

2. Open the source file `main.m` in an Xcode editor window.

3. Scroll down and find the call to the `NSApplicationMain` function.

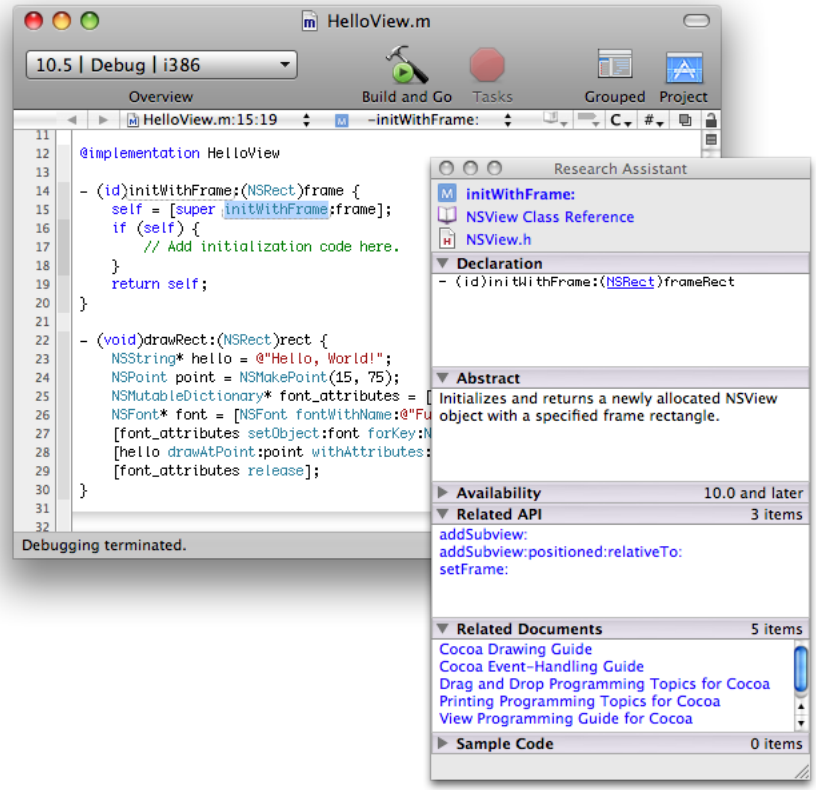
4. Option–double-click this function—that is, hold down the Option key and double-click the name of the function. Xcode opens the documentation window and displays reference information about this function.

The Research Assistant window (Help > Research Assistant) offers a concise view of essential reference documentation for a specific symbol. The information you can view in the Research Assistant includes a symbol's declaration, description, and availability information.

1. Open the Research Assistant window by choosing Help > Research Assistant.
2. Open the source file `HelloView.m` in an Xcode editor window.
3. Scroll down and find the call to the `initWithFrame:` method inside the brackets.
4. Click anywhere inside `initWithFrame:` or double-click to select it.

Xcode displays reference information about this method in the Research Assistant window, as shown in Figure 2-4.

Figure 2-4 Using the Research Assistant in Xcode



Searching for Relevant Documentation

The Full-Text Search mode in the documentation window lets you search the reference library for documents relevant to a particular search string.

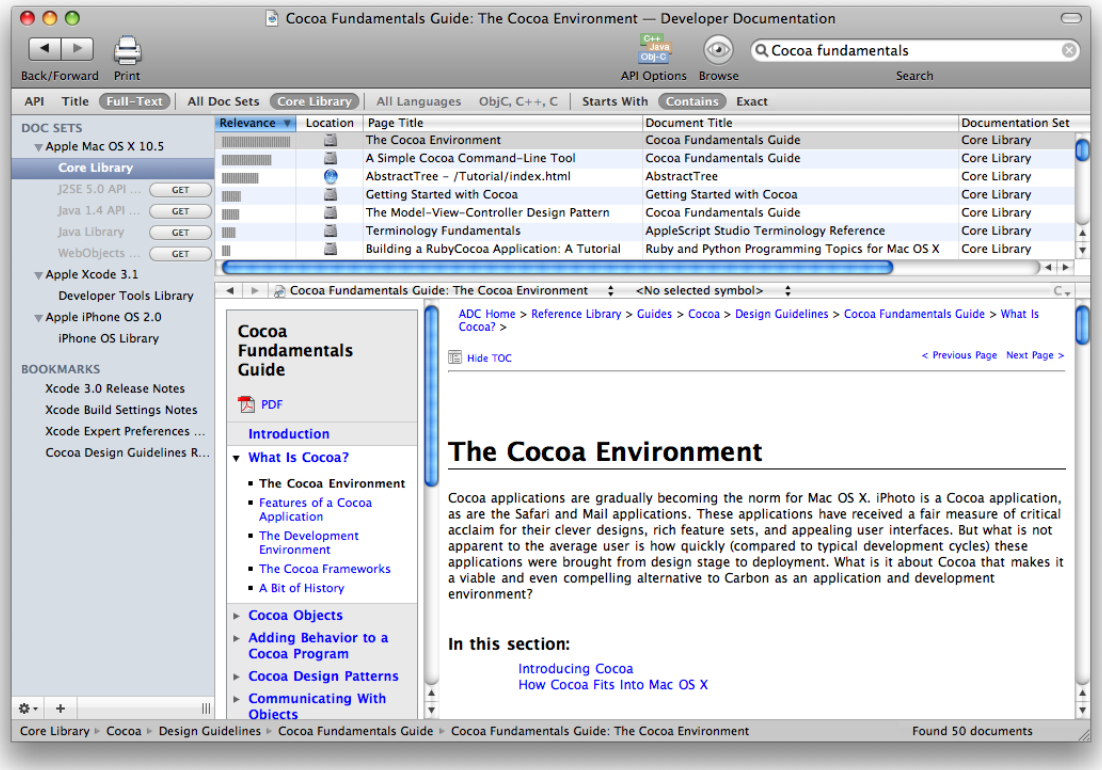
To search all technical documentation:

1. Launch Xcode and choose Help > Documentation. Xcode opens the window and displays a documentation page.

Note: After the first time you use the documentation window, Xcode remembers the last page you were viewing the next time you display the window.

2. Select the Core Library documentation set. This selection tells Xcode to search for Mac OS X documents.
3. In the search bar, click the Full-Text button and the Contains button.
4. Click in the search field, type the phrase `Cocoa fundamentals`, and press Return. Xcode finds documents that contain this phrase and lists them in order of relevance.
5. Select one of the items listed in the search results. If the item is located in your computer's local storage, Xcode displays the relevant page of documentation in the viewing pane, as shown in Figure 2-5.

Figure 2-5 Using the full-text search option in Xcode



Note: Reference Library items whose location icon is a blue globe instead of a hard disk are not located in your computer. When you click them, Xcode sends the corresponding URL to your web browser, which displays the webpage containing the item.

Following Links

The document in Figure 2-5 (page 34) contains a number of hypertext links to other pages in the library. By convention, links are displayed as blue text.

There are several ways to use these links:

1. Click any link in the viewing pane of the documentation window. Xcode displays the linked page in the same viewing pane. To return to the previous page, click the left arrow button above the viewing pane.
2. Command-click the same link. Now Xcode displays the linked page in a new Xcode window.
3. Option-click the link. This time, Xcode sends the page's URL to your web browser.

Finding Information in Headers

As you work on an Xcode project, sometimes it's useful to study the declarations and comments in a Mac OS X framework header. Typically, you already know the name of the header or the name of one of its symbols.

Finding Framework Headers

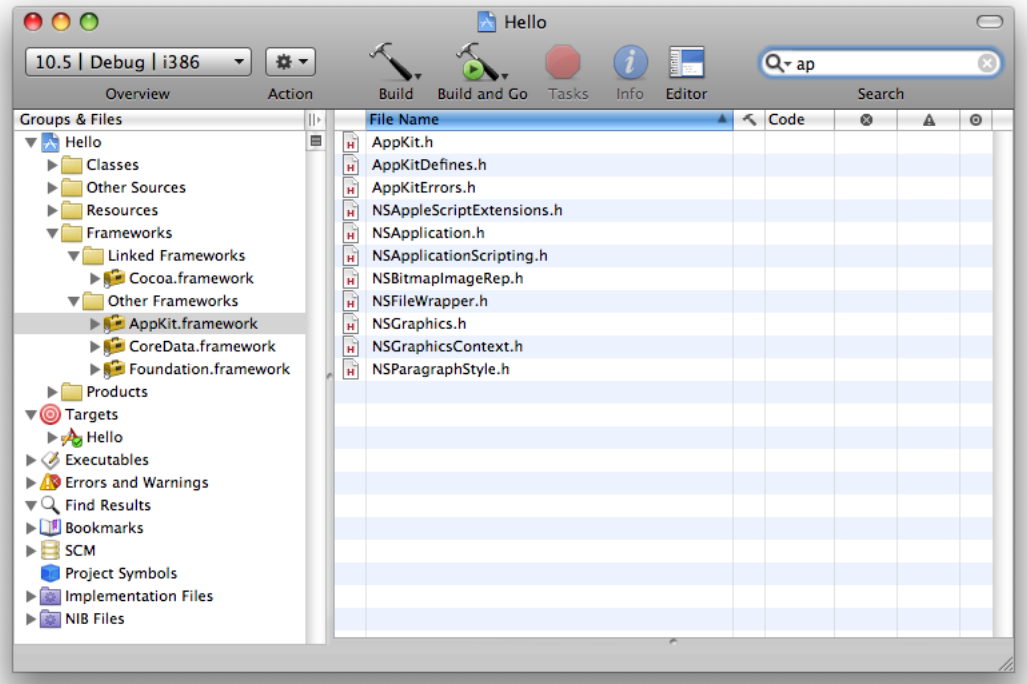
If you know the name of a header but not its location, you can use Xcode to locate the header in a framework included in your project.

For example, to locate a header in the AppKit framework:

1. In the project window, use the disclosure triangles to open the project group. The groups inside are called **source groups**. Source groups contain references to actual files somewhere on your hard disk.
2. Open up the source group named Frameworks, and select `AppKit.framework`. Xcode lists the header files contained in this framework in the detail view.

3. Click in the project window search field and slowly type a few letters—for example, "ap". Notice how Xcode filters the list of header files dynamically as you type each letter, as shown in Figure 2-6.

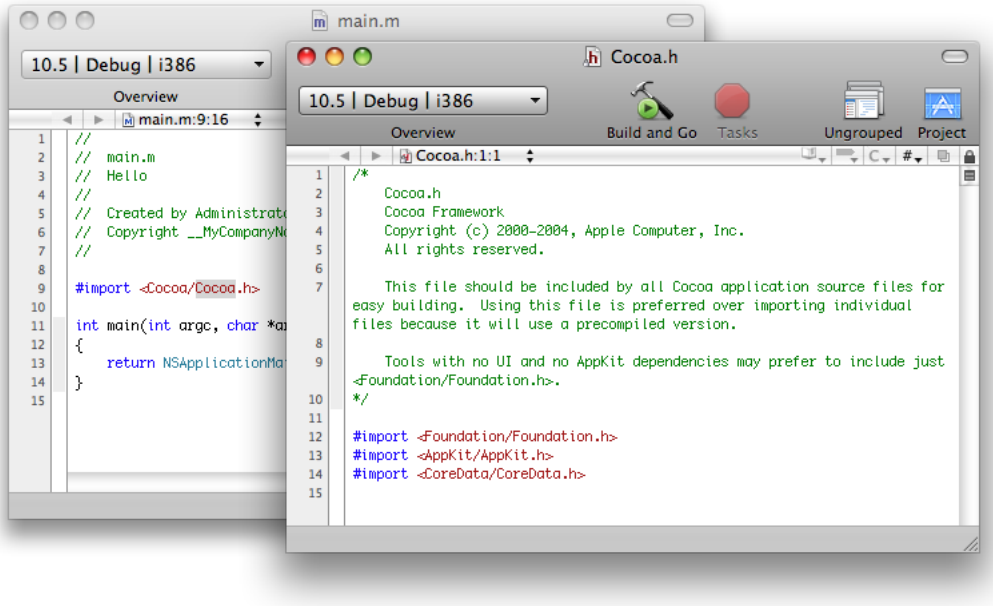
Figure 2-6 Searching for headers in a framework



Another way to find headers is to use Xcode's Open Quickly command. When the insertion point is on the name of a header in a source editor, you can press Shift–Command-D to bring up a source editor window with the corresponding header file. Xcode looks for the header file in the standard header locations.

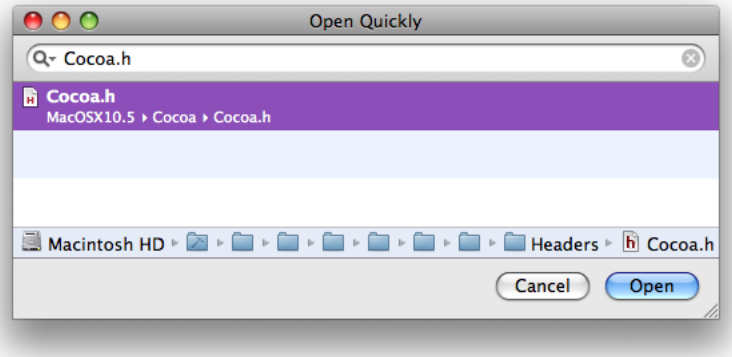
Figure 2-7 shows Open Quickly in action.

Figure 2-7 Using the Open Quickly command



If you're not editing a file or the insertion point is not over the name of a header, Xcode displays the Open Quickly dialog, as shown in Figure 2-8. You enter the name of header in the Path text field and click Open.

Figure 2-8 The Open Quickly dialog



Finding Symbol Declarations

In "Searching from a Text Editor Window" (page 32), you learned how to find the API documentation for any Apple-defined symbol in your source code. Using a similar technique, you can also find the declaration for a symbol.

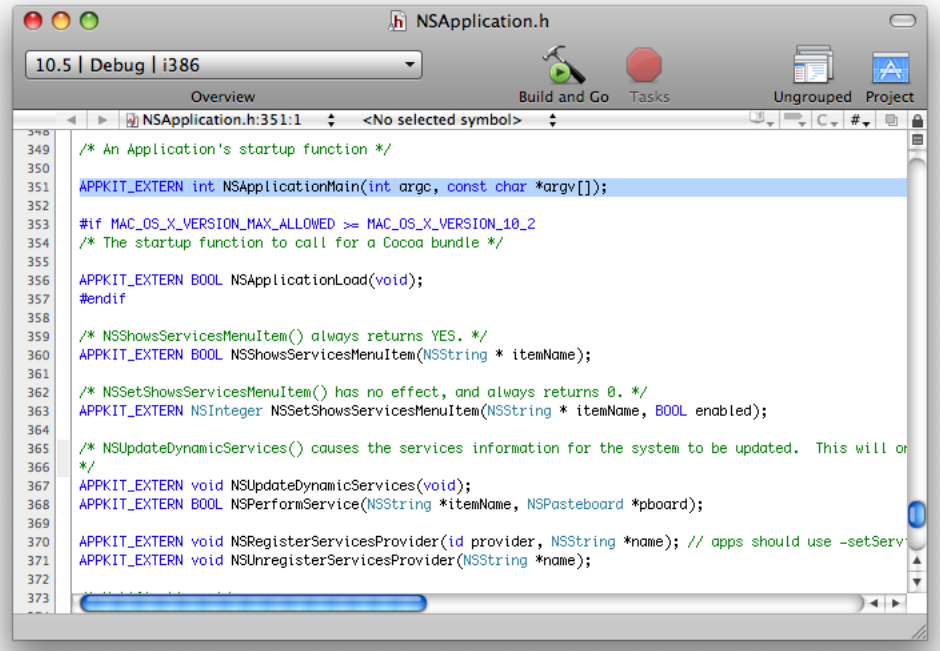
For example, to find the declaration for `NSApplicationMain`:

1. From the Hello project window, open the source file `main.m`.

2. Find the line where the function `NSApplicationMain` is called.

3. Command–double-click this function—that is, hold down the Command key and double-click the name of the function. Xcode opens the header file `NSApplication.h` and displays the declaration for `NSApplicationMain`, as shown in Figure 2-9.

Figure 2-9 Header search results



Summary

This tutorial provided an overview of the Xcode features that allow you to download and search for technical information. First, this tutorial described the process of downloading and updating your local copy of developer documentation. It showed how to search for conceptual documentation as well as API documentation from the documentation window and from a source editor window. Finally, this tutorial showed how to locate the header files in a framework, and how to find the header that declares a symbol used in a source file.

Designing a User Interface

Interface Builder is Apple's graphical editor for designing user interfaces. This tutorial shows how to use Interface Builder to design the user interface for a Cocoa application called Converter. After you're finished, you will incorporate your design into a working application.

To help you troubleshoot problems as you follow the tutorial, this document includes the completed Converter project in a companion archive named XcodeQuickTour_companion.zip.

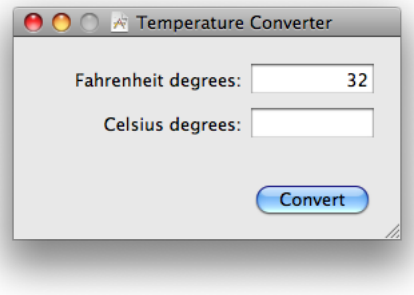
Note: If you're interested in learning how to build applications with Cocoa, you should also read *Cocoa Application Tutorial*.

Creating the Converter Interface

Converter is a simple application that converts temperatures from Fahrenheit degrees to Celsius degrees.

The user interface for Converter is displayed in a single window. Figure 3-1 shows what the application's user interface looks like after you've completed this tutorial.

Figure 3-1 The Converter user interface



To convert a temperature, you enter a value in the "Fahrenheit degrees" text field and press Return or click the Convert button. Converter displays the result in the "Celsius degrees" text field, which is read-only. To quit the application, you can either click the Close button or press Command-Q.

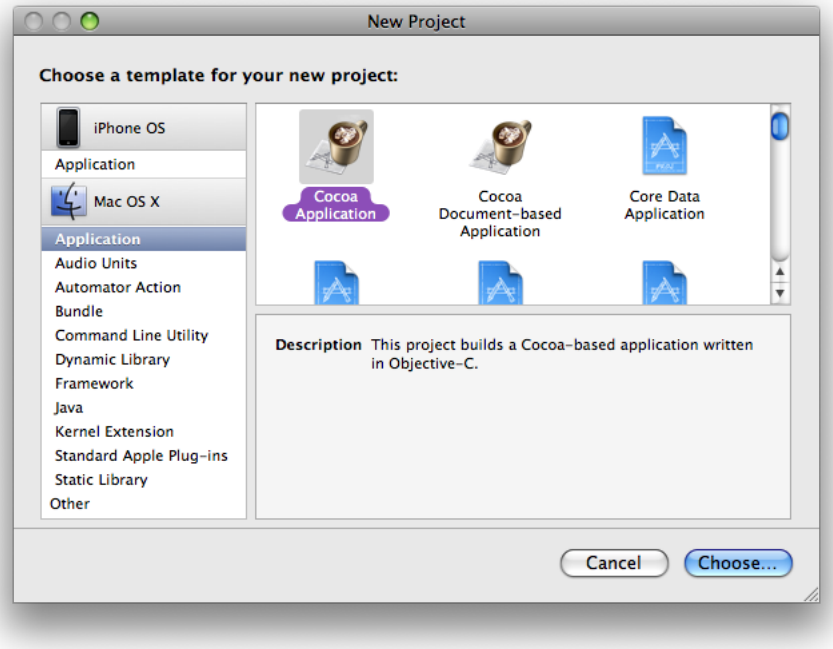
Getting Started

When you use Interface Builder to design a user interface, your work is saved in a file with the `.xib` or `.nib` extension. (A file of either type is referred to as a nib file.) At runtime, an application uses the contents of this file to construct and display its user interface, typically inside a window.

Xcode provides a template project that you can use to create a project for the Converter application. This template includes a nib file that defines a default user interface.

To create the Converter project and open its nib file:

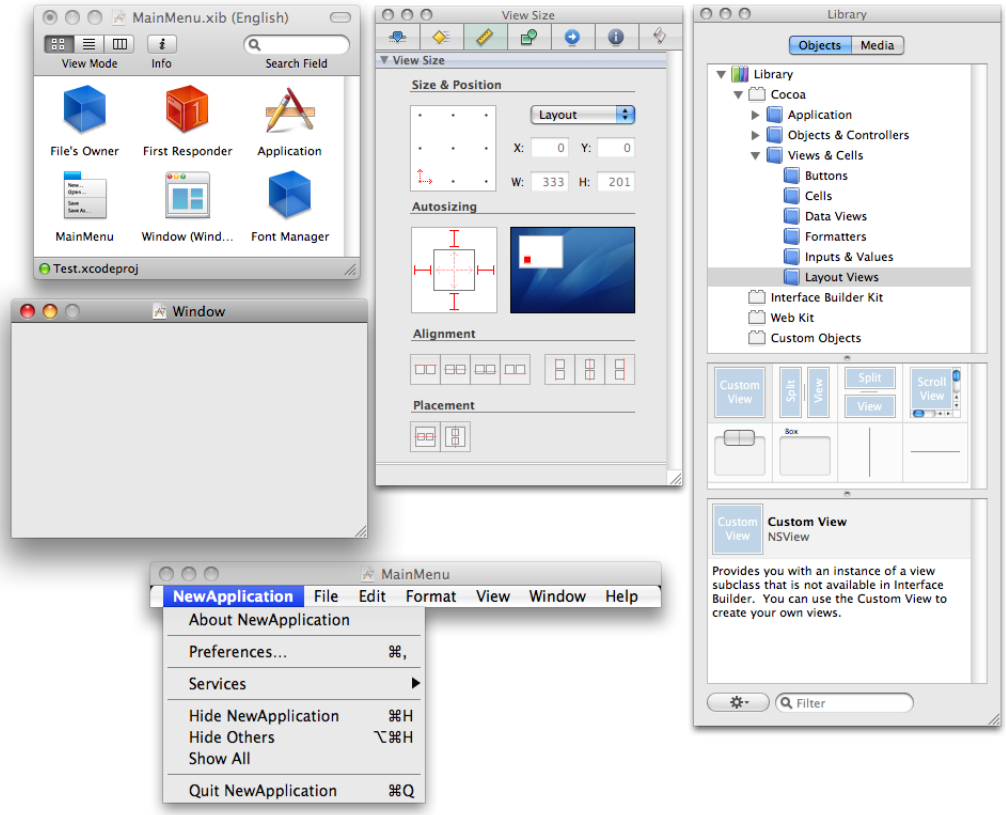
1. Launch Xcode and choose File > New Project (or press Command-Shift-N.) The New Project dialog appears.
2. Select the Cocoa Application template and click Choose.



3. Navigate to the location where you want Xcode to create the project folder.
4. Type `Converter` in the Save As field and click Save. Xcode creates the project and opens the project window.
5. In the Converter project window, select the Resources group.
6. In the project window's detail view, locate and double-click the `MainMenu.xib` file.

As shown in Figure 3-2, Interface Builder launches and presents a set of windows.

Figure 3-2 Editor windows in Interface Builder



Here are brief descriptions of the windows in Figure 3-2.

- **MainMenu.xib** represents the contents of the nib file. This window contains items titled MainMenu and Window that represent the application's menu bar and main window.
- **Window** is a visual representation of the main window. In it you place the controls that define the main window's user interface.
- **MainMenu** is a visual representation of the application's menu bar. You use this window to customize the menu bar.
- **View Size** is the size pane in the inspector window. You use the inspector to examine and change properties of any object in a nib file. The inspector window displays the properties of the currently selected object in multiple panes. The title of the inspector window changes based on the current context. To display this window, choose Tools > Inspector.
- **Library** contains objects, including controls and menu items, which you drag into other windows in the nib file to create the application's user interface. To display the library window, choose Tools > Library.

Setting the Window's Attributes

To set the attributes for the Converter window:

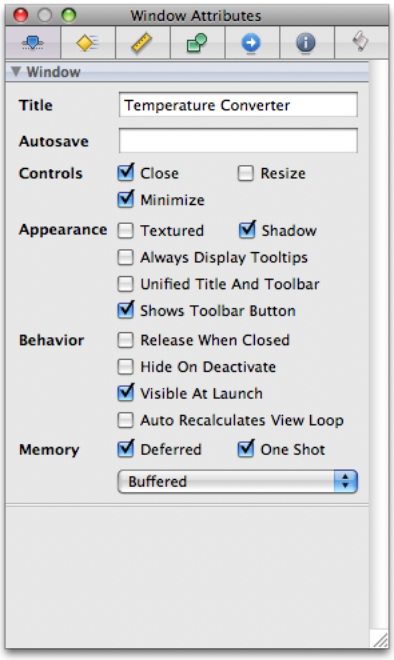
1. Click in the title bar of the main window to select it.

2. Choose Tools > Attributes Inspector (or press Command-1.) Interface Builder displays the attributes pane in the inspector window.

3. In the attributes pane, type `Temperature Converter` in the Title field and press Return to make the change appear in the window.

4. Deselect the Resize option—there's no need to support the window resizing control here.

   At this point, the window attributes should look like this:



5. Choose Tools > Size Inspector (or press Command-3.) Interface Builder displays the size pane in the inspector window.

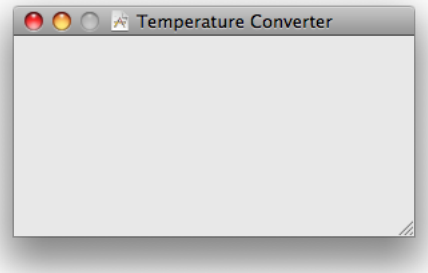
6. In the Content Size & Position group, set the window's width to 300 and its height to 150.

   There are two ways to resize a window in Interface Builder—by direct manipulation, or by specifying the exact dimensions as you have done here. These dimensions are not exactly correct, but they're a good first approximation. You set the exact dimensions later, after adding the window's user interface elements.

7. Set the initial position of the Converter window.

   The Apple Human Interface Guidelines says windows should open horizontally centered and positioned vertically so that the distance below the window is about two times the distance above the window. For now, simply drag the simulated window toward the center of the screen.

Your main window should now resemble this window:



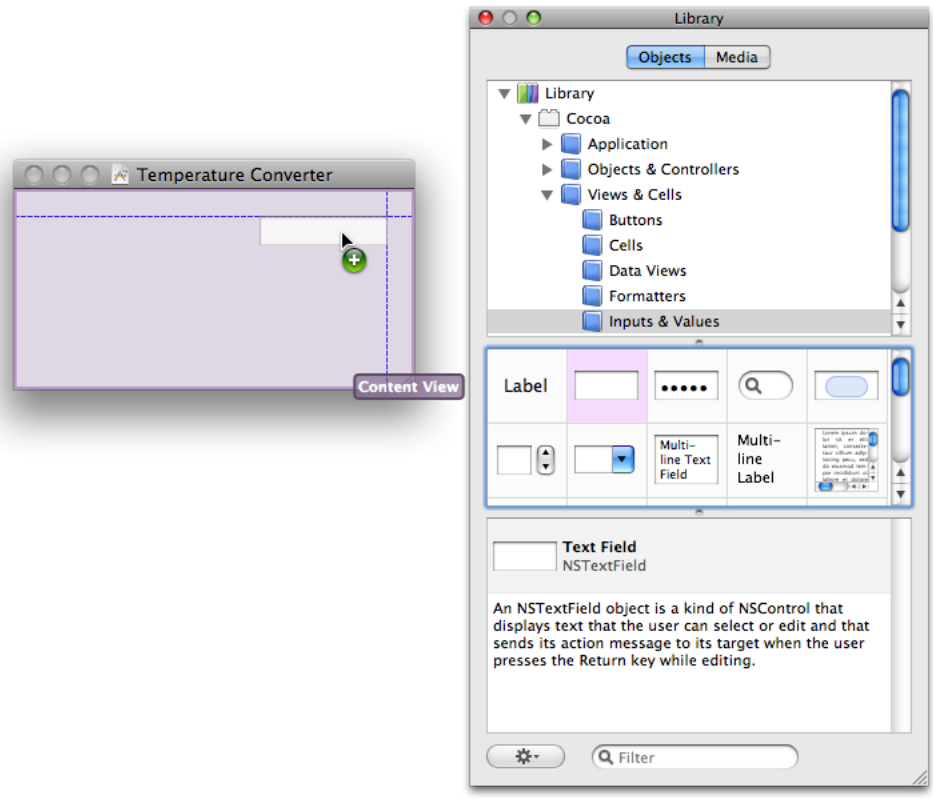
The inspector window is used throughout this tutorial, so it makes sense to leave this window open.

Adding the Fahrenheit Control

To add the Fahrenheit control:

1. In the library window, be sure that Objects is selected in the control at the top.
2. In the top pane, navigate to Library > Cocoa > Views & Cells and select Inputs & Values.

   You can view information about each control displayed in the middle pane by placing the pointer over it and clicking.
3. Drag an editable text field (NSTextField) from the palette to the main window.



Notice that Interface Builder helps you place objects according to the Aqua guidelines by displaying pop-up guides when an object is dragged close to the proper distance from neighboring objects or the edge of the window.

4. In the Converter window, select the editable text field and display the attributes inspector.
5. In the Title field, type `32` and press Return to apply the value.

   This serves as the default Fahrenheit temperature.
6. With the text field still selected, click the right-justified alignment tab.
7. Display the size inspector and set the width to 92.
8. Back in the main window, adjust the position of the text field as needed.

Adding the Celsius Control

Converter needs a second control to display the Celsius temperature, the same size as the first.

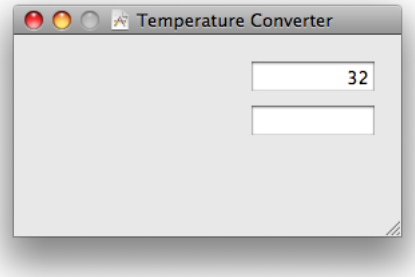
To duplicate the Fahrenheit control:

1. In the Converter window, select the editable text field.
2. Choose Edit > Duplicate (or press Command-D.)

   Interface Builder adds a new text field, slightly offset from the original.
3. Position the new text field below the original.

   Allow the guides to assist you by snapping the second field into place.
4. With the new field selected, display the attributes inspector and remove the title—a default Celsius temperature is not needed.
5. Deselect the Editable checkbox to make this a read-only control.

The Converter window should now resemble this window:



Adding Labels

Controls without labels would be confusing, so you're going to add these labels now. Cocoa uses static text fields for this purpose.

1. Drag a label from the library window to the Converter window.

   For now, you can place the label in the empty area below the temperature controls.

2. In the attributes inspector for the label, set the title to "Fahrenheit degrees:" and set the Alignment attribute to right-justified.

3. Use the size inspector to set the width of this label to 150.

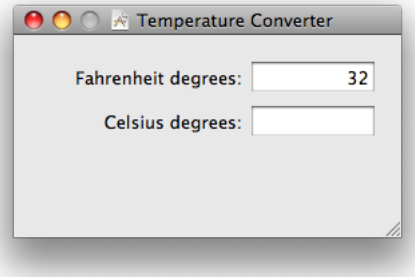
4. Back in the Converter window, position the label to the left of the first editable text field.

   Allow the guides to assist you by snapping it into place.

5. Make a duplicate label.

6. Set the title of this label to "Celsius degrees:" and position it to the left of the second editable text field.

The Converter window should now resemble this window:



Adding the Convert Button

The Converter application is designed to convert a Fahrenheit temperature to Celsius when the user presses Return or clicks the Convert button.

To add the Convert button:

1. Drag a push button from the library window (Cocoa > Views & Cells > Buttons) to the bottom-right portion of the Converter window.

2. With the button selected, use the attributes inspector to set the button's title to Convert.

3. In the Key Equiv. section of the inspector, click the query box and press the return key to make this a default button.

   In a user interface with a default button, pressing the return key simulates a button click.

4. With the button selected, use the size inspector to set the button width to 84.

5. Return to the main window, and align the button under the temperature fields.

   In order to make an attractive user interface, you must be able to visually align interface objects in rows and columns. Estimating correct alignment manually can be very difficult, and typing in x-y coordinates by hand is tedious and time consuming. Aligning Aqua user interface elements is made even more difficult because the objects have shadows and UI guideline metrics don't always take the shadows into account. Interface Builder uses visual guides to help you with object alignment.

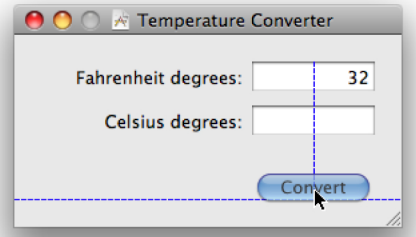
   a. Drag the button downward until a horizontal Aqua guide appears.

      This guide shows you that the button is the correct distance from the bottom of the window.

   b. Drag the button to the left or right until a vertical Aqua guide appears.

      This guide shows you that the button is centered with respect to the temperature fields.

   This figure illustrates how to position the Convert button with the aid of Aqua guides.



Finishing the Window Layout

The Converter user interface is almost complete. The finishing touch is to align the controls with respect to the top-left corner of the window and then resize the window. You keep using the automated Aqua guides, along with a few layout commands.

1. Select all the controls in the main window, either by dragging a rectangle around them or by pressing Command-A.

2. Drag the group (using any control as a handle) toward the top-left corner of the window, using the Aqua guides to help you find the proper position relative to the window's edges.

3. If necessary, move the button into position again under the temperature controls.

4. Locate the resize control at the bottom-right corner of the window. Drag the resize control inward, using the guides to give you the proper distance from the temperature fields (on the right) and the Convert button (on the bottom).

At this point, the main window should look like Figure 3-1 (page 39).

Testing the Interface

The Converter interface is now complete. Interface Builder provides a simulator that lets you test your interface without having to write one line of code.

To test your interface:

1. Choose File > Save to save your work.
2. Choose File > Simulate Interface (or press Command-R) to run the simulator.

   Note that the initial screen position you set in Interface Builder is used as the initial position for the Converter window in the simulator.
3. Enter a value in the Fahrenheit field, select the value, and cut and paste the value.
4. Make sure the Convert button is pulsating, indicating that it's the window's default button.
5. When you're finished testing the interface, press Command-Q to quit the simulator.

Congratulations! With Interface Builder, you have designed the user interface for a useful Cocoa application.

Implementing the Converter Application

The purpose of this section is to show you how to implement the Converter application, using the interface you have just designed. Because this tutorial is not designed to teach you Cocoa programming, it doesn't provide an explanation of what the code does.

Implementing the Model Class

Model objects contain special knowledge and expertise. They hold data and define the logic that manipulates that data. In the Converter application, the model class converts Fahrenheit degrees to Celsius degrees. Note that instances of a model class do not communicate directly with the user interface.

To create the two source files for the Converter model class:

1. In your Xcode project window, select the Classes group in the Groups & Files list.
2. Choose File > New File (or press Command-N.) The New File dialog appears.
3. In the list on the left, select the Mac OS X > Cocoa template.
4. In the list on the right, select Objective-C Class and click Next.
5. Name the file `Converter.m` and make sure the checkbox "Also create Converter.h" is selected.
6. Click Finish.

To add the source code that specifies the Converter class interface:

1. Open the source file `Converter.h`.
2. Add lines so that the code looks like that in Listing 3-1.

Listing 3-1 Converter.h

```
#import <Cocoa/Cocoa.h>

@interface Converter : NSObject {
    float fahrenheit;
}

@property(readwrite) float fahrenheit;

- (float) convertToCelsius;

@end
```

3. Save your changes to `Converter.h`.

To add the source code that specifies the Converter class implementation:

1. Open the source file `Converter.m`.
2. Add lines so that the code looks like that in Listing 3-2.

Listing 3-2 Converter.m

```
#import "Converter.h"

@implementation Converter

@synthesize fahrenheit;

- (float) convertToCelsius {
    return (self.fahrenheit - 32.0) * 0.555556;
}

@end
```

3. Save your changes to `Converter.m`.

Implementing the Controller Class

The primary purpose of the controller class is to communicate between the model class and the user interface you created earlier in this chapter. In Converter, the controller class is also a delegate for the application class.

To create the two source files for the controller class:

1. In your Xcode project window, select the Classes group in the Groups & Files list.
2. Choose File > New File (or press Command-N.) The New File dialog appears.

3. In the list on the left, select the Mac OS X > Cocoa template.
4. In the list on the right, select Objective-C Class and click Next.
5. Name the file `Controller.m` and make sure the checkbox "Also create Controller.h" is selected.
6. Click Finish.

To add the source code that specifies the Controller class interface:

1. Open the source file `Controller.h`.
2. Add lines so that the code looks like that in Listing 3-3.

Listing 3-3 Controller.h

```
#import <Cocoa/Cocoa.h>

@interface Controller : NSObject {
    IBOutlet NSTextField *fahrenheitField;
    IBOutlet NSTextField *celsiusField;
}
- (IBAction)convert:(id)sender;

@end
```

3. Save your changes to `Controller.h`.

To add the source code that specifies the Controller class implementation:

1. Open the source file `Controller.m`.
2. Add lines so that the code looks like that in Listing 3-4.

Listing 3-4 Controller.m

```
#import "Controller.h"
#import "Converter.h"

@implementation Controller

- (IBAction) convert:(id)sender
{
    Converter *converter = [[Converter alloc] init];
    [converter setFahrenheit:[fahrenheitField floatValue]];
    float celsius = [converter convertToCelsius];
    [celsiusField setStringValue:[NSString stringWithFormat:@"%.2f", celsius]];
    [fahrenheitField selectText:self];
    [converter release];
}

- (BOOL) applicationShouldTerminateAfterLastWindowClosed:(NSApplication
*)theApplication
{
    return YES;
```

```
}

@end
```

3. Save your changes to `Controller.m`.

Adding a Controller to Your Nib File

In order to add connections between the controller and other objects in your design, you need to create an instance of the controller class in your nib file.

1. In Interface Builder, choose File > Open Recent to open the `MainMenu.xib` file in the Converter project.
2. In the library window, select Cocoa > Objects & Controllers.
3. Drag an instance of `NSObject` into the `MainMenu.xib` window.
4. Select the new object and press Command-6 to open the identity inspector.
5. In the Class field, type `Controller`.
6. Save your changes to the nib file.

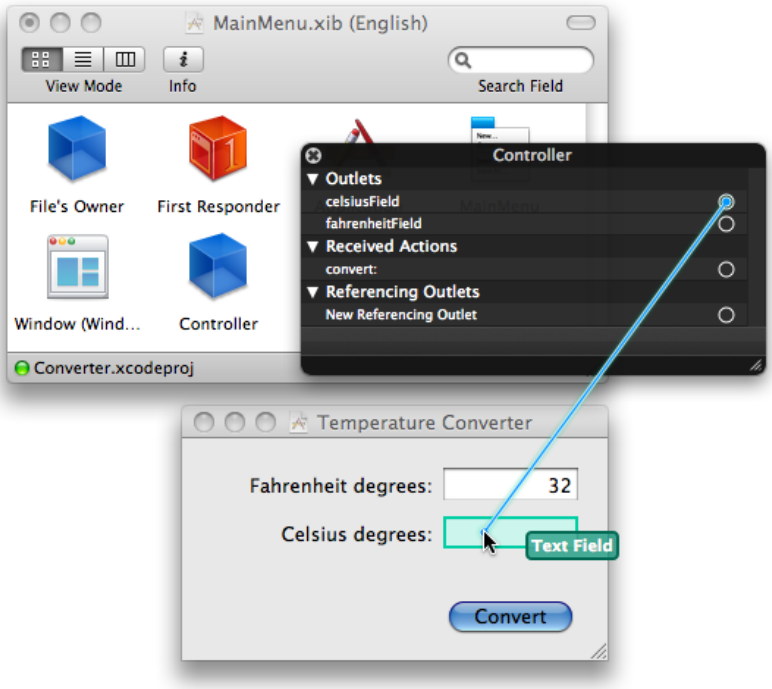
Adding the Controller Connections

Interface Builder not only helps you design your user interface, it makes it possible to connect the properties of the various objects in your design. Several objects in the Converter interface need to be connected to properties in the controller class.

To add the connections:

1. In the `MainMenu.xib` window, Control-click the controller object. A connections panel appears.

2. Drag from the circle to the right of the `celsiusField` to the Celsius degrees field in the Temperature Converter window, as shown below.



When you release the mouse, the connection is established and the circle is filled.

3. Using the same technique, add a connection from the `fahrenheitField` outlet to the Fahrenheit degrees field.

4. Add a connection from the `convert:` action to the Convert push button.

5. Add a connection from New Referencing Outlet to the File's Owner object. When you release the mouse, an outlet named `delegate` appears. Select this outlet.

6. Close the connections panel and save your changes to the nib file.

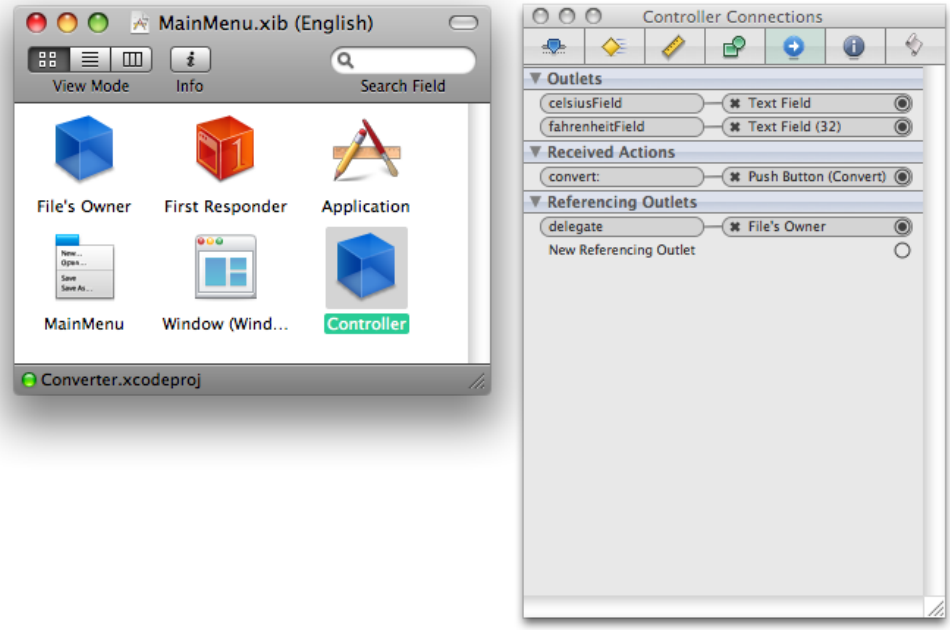
Inspecting the Controller Connections

With Interface Builder's connections inspector, you can examine and verify the connections in a nib file.

1. With the Controller object still selected, press Command-5 to display the connections inspector.

2. Verify that there are four connections: two outlets for the text fields, a received action for the Convert button, and an application delegate outlet. Figure 3-3 shows these connections in the inspector.

Figure 3-3 Connections in Interface Builder



3. Now select the Convert button in the main window.

   The connections inspector shows a sent action for the Controller object. This action represents the other end of the received action shown in Figure 3-3. The two text fields and the File's Owner object have similar complementary connections.

Building and Running the Application

To build and run the Converter application:

1. In the Xcode project window, click the Build toolbar item (or press Command-B.)

   The status bar at the bottom of the project window indicates the status of the build. When Xcode finishes—and encounters no errors along the way—it displays "Build succeeded" in the status bar. If there are errors, however, you need to correct them and start the build process again.

2. Click the Build and Go toolbar item (or press Command-Return.)

3. After the Converter application launches, enter the Fahrenheit degrees you wish to convert and click the Convert button. A value should appear in the Celsius degrees field.

4. Enter a different Fahrenheit value and press the Return key. This is equivalent to clicking the Convert button.

5. When you're finished using the application, close the Converter window. Notice that the application quits.

What's Next?

To make Converter a solid Mac OS X application, clearly there's more work to be done. If you're interested in Cocoa programming, here are a few ideas to improve and extend this application:

- Edit the menu bar using Interface Builder. The default Cocoa application menu bar contains items that don't apply here and should be removed.
- Add range checking for the Fahrenheit temperature control. Absolute zero is around –459.6 degrees Fahrenheit.
- Add conversion from Celsius to Fahrenheit.

Summary

The Converter tutorial showed you how to implement the user interface for a simple Cocoa application using Interface Builder. First, the tutorial described how to lay out a window's user interface elements. Next, the tutorial demonstrated how to test an application's user interface before implementing its logic. Then, the tutorial helped you make the necessary connections between user interface objects and class outlets and action methods. Finally, you built and ran the Converter application to verify that the user interface was constructed correctly.

Using Fix and Continue

Fix and Continue makes it possible to modify an executable image at debug time. You can see the results of your modification without interrupting or restarting the debugging session.

This feature is especially useful when it takes a lot of time to reach a particular state of execution while debugging your application. Rather than recompiling your project and starting a new debugging session, you can make minor changes to your code, fix your executable image, and see the results of your changes immediately. For detailed information on Fix and Continue, see Modifying Running Code in *Xcode Debugging Guide*.

This tutorial uses a sample project named Sketch, which implements a simple drawing program built using the Cocoa framework. You're going to change the way Sketch draws shapes by changing the source code and using the Fix command in Xcode to patch the running program.

You don't need to be familiar with Objective-C to work through this tutorial.

Configuring the Project

To configure the Sketch project to use the Fix command:

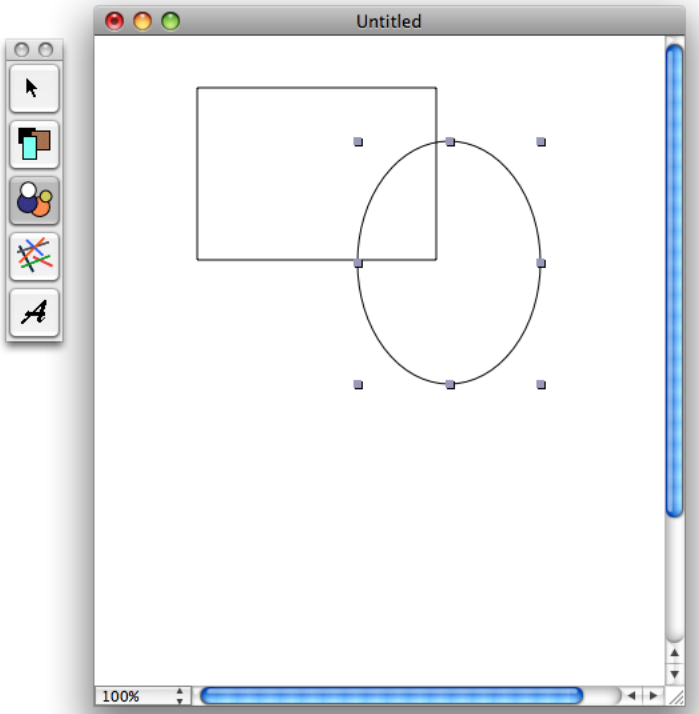
1. Make a copy of the Sketch project, which is located in `/Developer/Examples/AppKit`. To avoid confusion, place your copy somewhere inside your home directory.
2. In the Finder, drag your copy of the Sketch project folder onto the Xcode icon in the Dock. Notice that Xcode finds and opens the project file inside this folder.
3. Make sure the active build configuration is Debug. From the Project menu, choose Set Active Build Configuration > Debug.
4. From the Build menu, choose Build (Command-B) to build the target.

Patching the Running Application

1. To start a debugging session, choose Run > Debug (Option–Command-Y).

2. Click in the Sketch drawing window, select a drawing tool (rectangle or circle), and draw a few objects.

Figure 4-1 The default drawing behavior of Sketch



3. Return to the project window and select the Sketch group.
4. Open the source file named `SKTGraphic.m`.
5. In the source editor, locate the `-init` method. It looks something like this:

```
- (id)init {
    self = [super init];
    if (self) {
        _bounds = NSZeroRect;
        _isDrawingFill = NO;
        _fillColor = [[NSColor whiteColor] retain];
        _isDrawingStroke = YES;
        _strokeColor = [[NSColor blackColor] retain];
        _strokeWidth = 1.0f;
    }
    return self;
}
```

6. Change some of the values.

   For example, change the following lines:

```
        _isDrawingFill = NO;
```

```
_fillColor = [[NSColor whiteColor] retain];
_strokeWidth = 1.0f;
```

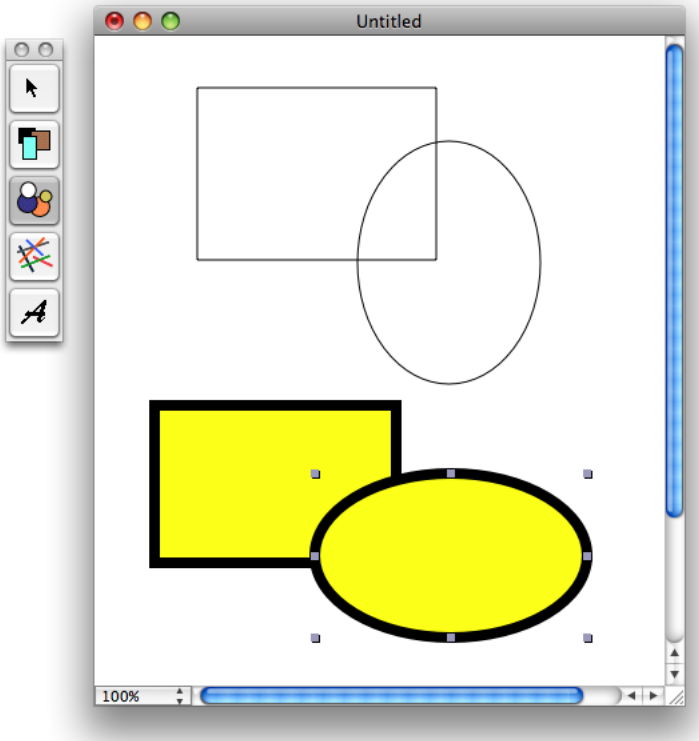
to:

```
_isDrawingFill = YES;
_fillColor = [[NSColor yellowColor] retain];
_strokeWidth = 8.0f;
```

7. Save your changes.
8. Choose Run > Fix to modify the running program.
9. Return to the Sketch drawing window and draw some more objects.

   You should see a dramatic change in the appearance of these new objects, as illustrated in Figure 4-2.

Figure 4-2 The new drawing behavior of Sketch



Summary

This tutorial showed how to use Fix and Continue to patch changes into a running application.

Document Revision History

This table describes the changes to *Xcode Quick Tour for Mac OS X*.

Date	Notes
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.
	Updated the introduction and "Creating a Project" (page 9) to specify the development environment required to successfully complete the tasks described in this document.
2005-09-08	Updated introduction to clarify how to install Xcode Tools.
	Corrected screenshot in "Attributes of the Converter window" (page 42).
2005-06-06	Updated for Mac OS X v10.4. Changed title from "A Quick Tour of Xcode."
	Added information on code completion, build configurations, breakpoint actions, and Open Quickly in "Creating a Project" (page 9).
	Added ADC Reference Library–update information in "Finding Technical Information" (page 29).
2003-08-28	First public version of this document.
2003-06-20	Released as a preliminary document for WWDC 2003.

Document Revision History