

---

# A Quick Tour of Xcode



2003-08-28



Apple Computer, Inc.  
© 2003 Apple Computer, Inc.  
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Computer, Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

The Apple logo is a trademark of Apple Computer, Inc.

Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-labeled or Apple-licensed computers.

Every effort has been made to ensure that the information in this document is accurate. Apple is not responsible for typographical errors.

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

Apple, the Apple logo, Aqua, Carbon, Cocoa, Mac, and Mac OS are trademarks of Apple Computer, Inc., registered in the United States and other countries.

Finder, Safari, and Xcode are trademarks of Apple Computer, Inc.

Objective-C is a trademark of NeXT Software, Inc.

Times is a trademark of Heidelberger Druckmaschinen AG, available from Linotype Library GmbH.

Simultaneously published in the United States and Canada.

**Even though Apple has reviewed this manual, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS MANUAL, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS MANUAL IS SOLD "AS IS," AND YOU, THE PURCHASER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.**

**IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS MANUAL, even if advised of the possibility of such damages.**

**THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.**

**Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.**

# Contents

**Introduction**      [Introduction to A Quick Tour of Xcode](#) 7

---

[See Also](#) 7

**Chapter 1**      [Creating a Project](#) 9

---

- [Creating an Xcode Project](#) 9
- [The Project Window](#) 11
  - [The Toolbar and Status Bar](#) 11
  - [Groups & Files](#) 12
- [Editing Project Files](#) 13
  - [Using the Source Editor](#) 13
  - [Using Interface Builder](#) 14
- [Building the Application](#) 15
  - [Configuring Build Settings](#) 15
  - [Building a Target](#) 15
- [Running the Application](#) 16
  - [Compile-Time Errors](#) 16
  - [Run-time Debugging](#) 17
- [Summary](#) 19

**Chapter 2**      [Finding Technical Information](#) 21

---

- [Finding Technical Documentation](#) 21
  - [Searching For All Relevant Documentation](#) 21
  - [Following Links](#) 23
  - [Searching For API Documentation](#) 23
- [Finding Information in Headers](#) 26
  - [Finding Framework Headers](#) 26
  - [Finding Symbol Declarations](#) 27
- [Summary](#) 27

**Chapter 3**      [Designing a User Interface](#) 29

---

- [Creating the Converter Interface](#) 29
  - [Getting Started](#) 30
  - [Setting the Window's Attributes](#) 31
  - [Adding the Fahrenheit Control](#) 32

- Adding the Celsius Control 33
- Adding Static Text Labels 34
- Adding the Convert Button 35
- Aqua Layout and Object Alignment 35
- Finishing the Window Layout 36
- Testing the Interface 37
- Implementing the Converter Application 37
- Summary 38

**Chapter 4**      **Using Fix and Continue** 39

---

- Configuring the Project 39
- Fixing the Running Application 40
- Summary 43

**Appendix A**      **Converter Source Code** 45

---

- Revision History 49

---

# Tables, Figures, and Listings

## Chapter 1      Creating a Project    9

---

- Figure 1-1    The Xcode application icon    9
- Figure 1-2    The New Project window    10
- Figure 1-3    An Xcode project window    11
- Figure 1-4    The project window toolbar    11
- Figure 1-5    Info window in Interface Builder    14
- Figure 1-6    Main window for the Hello application    16
- Figure 1-7    Error and warning messages    17
- Figure 1-8    Setting a breakpoint    18
- Figure 1-9    The Debug window    18

## Chapter 2      Finding Technical Information    21

---

- Figure 2-1    Using the full-text search option in Xcode    22
- Figure 2-2    Full-text search with two search terms    23
- Figure 2-3    Using the API-reference search option in Xcode    24
- Figure 2-4    Partial symbol search    25
- Figure 2-5    Searching for headers in a framework group    26
- Figure 2-6    Header search results    27

## Chapter 3      Designing a User Interface    29

---

- Figure 3-1    The Converter user interface    29
- Figure 3-2    Editing windows in Interface Builder    30
- Figure 3-3    Attributes of the Converter window    31
- Figure 3-4    Converter window before adding controls    32
- Figure 3-5    Adding an editable text field    32
- Figure 3-6    Converter window after adding temperature controls    34
- Figure 3-7    Converter window after adding static text    34
- Figure 3-8    Measuring distances in Interface Builder    35
- Figure 3-9    Layout rectangles in Interface Builder    36

## Chapter 4      Using Fix and Continue    39

---

- Figure 4-1    Choosing the native build target    40
- Figure 4-2    The default drawing behavior of Sketch    41
- Figure 4-3    The new drawing behavior of Sketch    42

Table 4-1 Types of build targets 39

**Appendix A** Converter Source Code 45

---

Listing A-1 Converter source code 45

# Introduction to A Quick Tour of Xcode

---

Xcode is the developer tools package for Mac OS X. This package includes an integrated suite of software development tools, 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 refers to the Xcode application.)

This quick tour gives you a hands-on introduction, in the form of four short tutorials, to the Xcode application and some of its companion tools.

The tour starts with “[Creating a Project](#)” (page 9), an introduction to some of the basic features in Xcode.

“[Finding Technical Information](#)” (page 21) shows how to quickly find and display technical information in Apple’s developer documentation library.

The third stop on the tour is “[Designing a User Interface](#)” (page 29), an introduction to Interface Builder. You’ll learn how to use Interface Builder to design a user interface for a sample Carbon application.

The Xcode application has many advanced features that aren’t covered here, but one feature is so interesting it’s worth a look. The tour finishes with “[Using Fix and Continue](#)” (page 39), a short tutorial that uses the Fix command to modify a running application during a debugging session.

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.

## See Also

---

- For information about all aspects of software development in Mac OS X, visit the Apple Developer Connection website at <http://developer.apple.com/>.
- Documentation about features in Xcode and Interface Builder is available in their respective Help menus.
- For an overview of the software development tools in Xcode, see Mac OS X Development Tools Overview.

# I N T R O D U C T I O N

## Introduction to A Quick Tour of Xcode

- For tips on converting CodeWarrior projects into Xcode projects, see *Moving Projects From CodeWarrior to Xcode*.
- To learn more about the capabilities and limitations of the Fix command in Xcode, see the programming topic *Fix and Continue*.
- 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 the *Aqua 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.

In this short tutorial, you will create an Xcode project for a Carbon application called Hello that prints “Hello, World!” inside a window. Along the way, you’ll get a chance to explore some of the basic features in Xcode.

## 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:

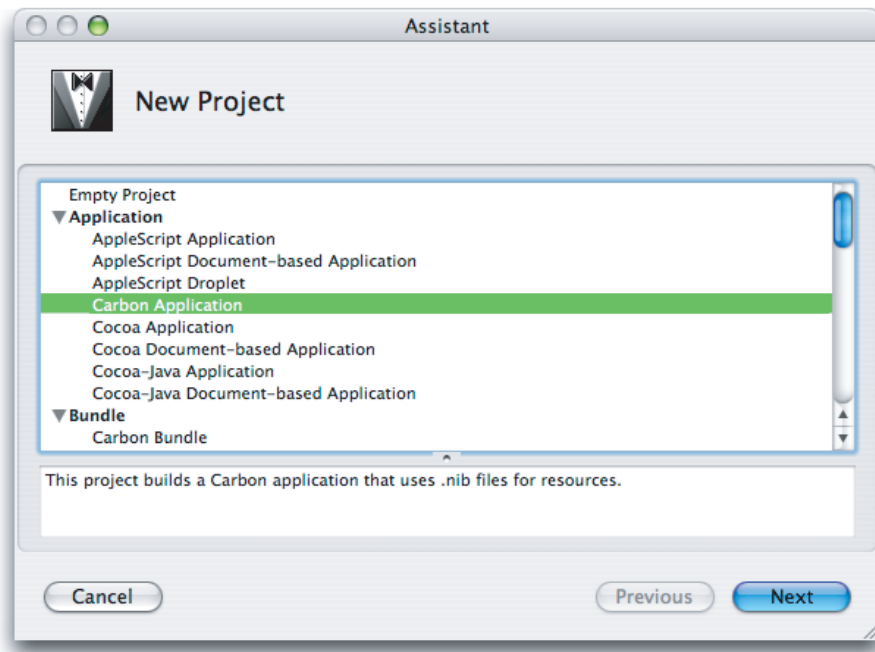
1. Find Xcode in the `/Developer/Applications/` directory, and double-click its icon (see Figure 1-1). The first time you use Xcode, it asks a few setup questions. The default values should work for the majority of users.

**Figure 1-1** The Xcode application icon

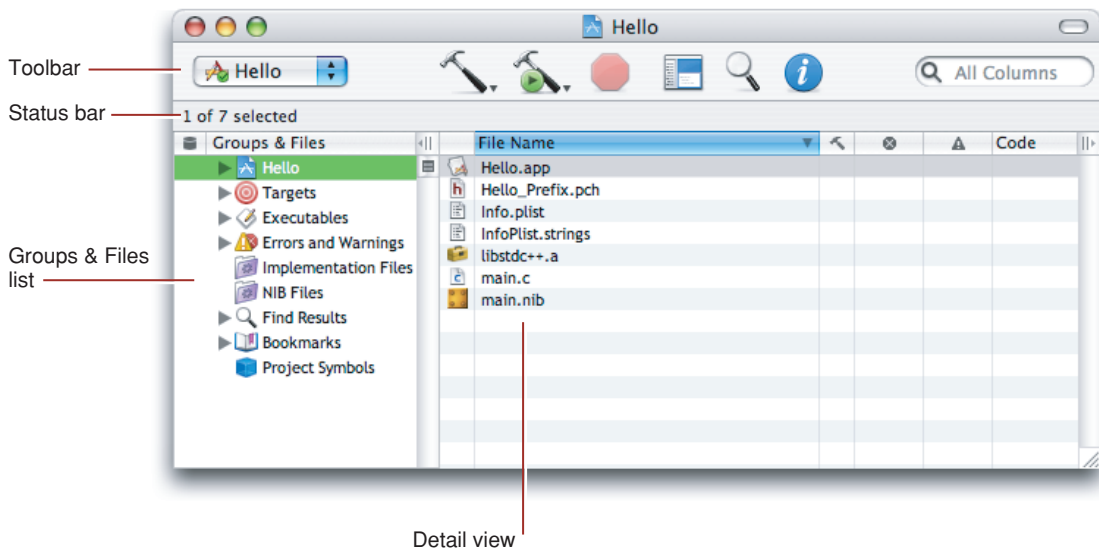


2. Choose `File > New Project` to display the New Project window. If you’re curious, browse through the list of templates to see the variety of software products Xcode can build.
3. Select the Carbon Application template as shown in Figure 1-2, and click Next.

Figure 1-2 The New Project window



4. Type the name `Hello` in the Project Name field.
5. Click Choose and navigate to the location where you want the Xcode application to create the project folder.
6. Click Finish. The Xcode application creates the project and displays the project window, as shown in Figure 1-3.

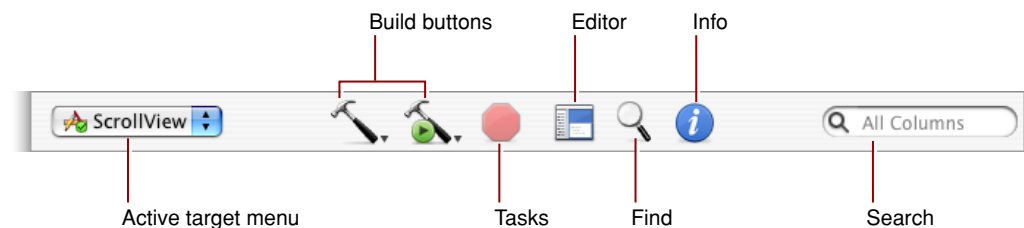
**Figure 1-3** An Xcode project window

## 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 Target pop-up menu selects the active build target.
- Build buttons initiate actions such as building, cleaning, running, and debugging.

**Note:** Toolbar buttons with drop-down triangles have additional commands associated with them. Pressing one of these buttons allows you to choose a different command from a pop-up menu.

- The Tasks button allows you to stop any operation in progress.
- The Editor button shows or hides the source editor in the project window.
- The Find button opens a Find dialog to search the content in your project.
- The Info button opens an Info window to examine and edit groups, files, and targets in your project.
- The Search field filters the items currently displayed in the detail view.

Located directly below the toolbar is the status bar, which displays status messages for the project. For example, in [Figure 1-3](#) (page 11) the status bar shows the number of selected items in the detail view.

## 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 you define your own groups.

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 “smart” groups that show items with a particular characteristic. The items in a dynamic group can change as you perform various actions in your project.

Here are some of the basic groups:

- The project group organizes all of the components needed to build your product. This group is always listed first, and it's 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

---

In this section, you'll modify the behavior of the application to print "Hello, World!" in its main window.

To implement this new behavior, you will:

- Add source code for a function that draws the textual greeting.
- Add source code that installs this function as a window event handler.
- Modify an attribute of the main window using Interface Builder.

### Using the Source Editor

---

Xcode has a built-in source 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. Select the project group in the Groups & Files list, as shown in [Figure 1-3](#) (page 11).
2. Select `main.c` in the detail view on the right.
3. Do one of the following:
  - Double-click to edit the file in a new editor window.
  - Click the Editor button, shown in [Figure 1-4](#) (page 11), to edit the file inside the project window.
4. Just before the main function, insert the following lines:

```
static OSStatus PrintHello (
    EventHandlerCallRef handler, EventRef event, void *data)
{
    WindowRef window = (WindowRef) data;
    Rect bounds;

    GetWindowPortBounds (window, &bounds);
    EraseRect (&bounds);
    InsetRect (&bounds, 12, 12);
    TextFont (FMGetFontFamilyFromName ("\pTimes"));
    TextSize (48);
    TextFace (italic);
    TXNDrawCFStringTextBox (CFSTR("Hello, world!"), &bounds, NULL, NULL);
    return noErr;
}
```

5. Insert the following lines just before the call to `ShowWindow` inside the main function:

```
EventTypeSpec eventSpec =
    { kEventClassWindow, kEventWindowDrawContent };
```

```
InstallWindowEventHandler (window, NewEventHandlerUPP(PrintHello),
    1, &eventSpec, (void *) window, NULL );
```

6. Save your changes by choosing File > Save (Command-S).

## Using Interface Builder

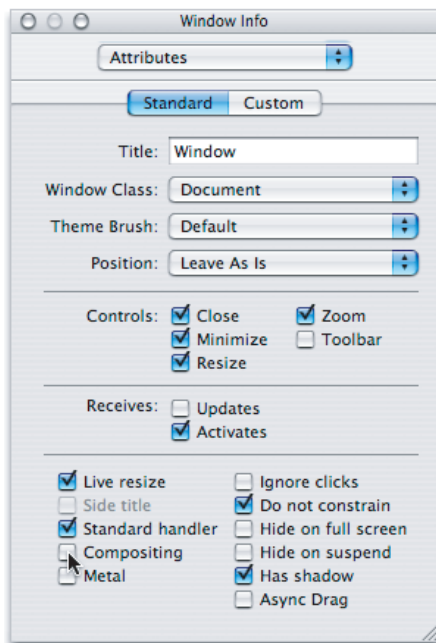
---

The final task before building and running the Hello application is modifying several attributes of the main window. You will use Interface Builder—Apple’s user interface design tool—to do this.

To make these changes:

1. Select the project group in the Groups & Files list, as shown in [Figure 1-3](#) (page 11).
2. Select the file `main.nib`, and double-click to open the file. Interface Builder launches and displays several editing windows, which represent different interface components and editing tasks.
3. Select the main window by clicking anywhere inside it.
4. From the Tools menu, choose Show Info (Shift-Command-I) to open the Window Info window, as shown in [Figure 1-5](#).

**Figure 1-5** Info window in Interface Builder



5. Edit the Title attribute. For example, change the window’s title to “Hello” or “Main Window”. This text appears in the title bar when the window opens.
6. Uncheck the Compositing attribute. To learn what compositing means in this context, see *Introducing HView*.

7. Choose File > Save (Command-S) to save the modified nib file.

For now, you're finished using Interface Builder. In the tutorial in [Chapter 4, "Designing a User Interface"](#), you will gain more experience using this important tool.

## Building the Application

---

Now it's time to build your application.

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

One of these 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.

## Configuring Build Settings

---

The build system in Xcode handles the complex process of creating a finished software product, based on rules and settings specified in the project. In the Hello project, you can simply use the defaults inherited from the Carbon Application template.

**Note:** Learning how to configure the build settings in a project goes beyond the scope of this tutorial. For information about these and other topics, choose Help > Xcode Help.

## Building a Target

---

Xcode requires that you specify a target to build. A target represents a single software product, and Xcode supports multiple targets in a project. The Hello project has a single target that specifies how to build a Carbon application.

Building this target is a simple two-step procedure:

1. From the popup menu in the upper-left corner of the project window, select the target named Hello. (Because this project has only one target, it's selected by default.)
2. Do one of the following:
  - From the Build menu, choose Build (Command-B).
  - Click the Build button in the toolbar (the left build button in [Figure 1-4](#) (page 11)).

When Xcode finishes building the target—and encounters no errors along the way—it displays “Build succeeded” in the status bar of the project window.

## Running the Application

---

Now you have an application that’s ready to run. Xcode installs the application bundle in a location specified in your project’s settings—in this case, inside your project folder.

To verify that the application runs:

1. Choose Debug > Run Executable (Command-Option-R).
2. Verify that the application opens a window, displays “Hello,World!” and waits for user interaction.

**Figure 1-6** Main window for the Hello application



3. Close the window by clicking the window’s close button or typing Command-W. Notice that the application is left running without a window. (If you’re curious, [Listing A-1](#) (page 45) includes source code that shows how to handle this event and quit the application.)
4. Type Command-Q to quit.

## Compile-Time Errors

---

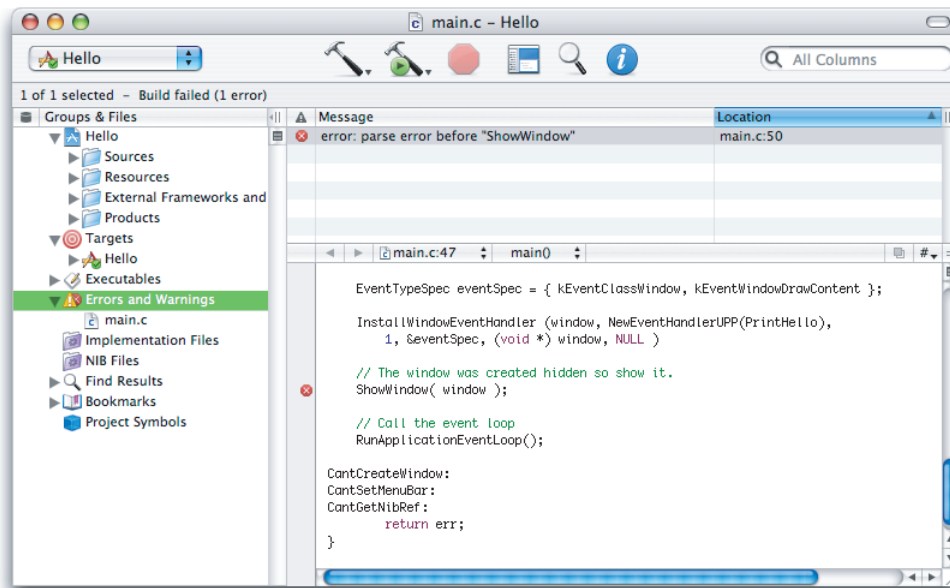
Of course, projects are rarely flawless from the start. By introducing a mistake into the source code for the Hello project, you can demonstrate the error-checking features in Xcode.

To demonstrate error checking:



1. Open `main.c` in either the project window or a separate editor window.
2. Delete a semicolon inside the body of a function, creating a syntax error.
3. Choose File > Save (Command-S) to save the modified file.
4. Choose Build > Build (Command-B). As expected, this time the build fails.
5. To view the error and warning messages in the project window, select Errors and Warnings in the Groups & Files list, as shown in Figure 1-7. The messages are displayed in the detail view to the right, and in the gutter of the editor window.

**Figure 1-7** Error and warning messages



6. Fix the error in the code, save, and build again. If the build succeeds, Xcode creates a new version of the product. Notice that the error messages from the previous build have been cleared.

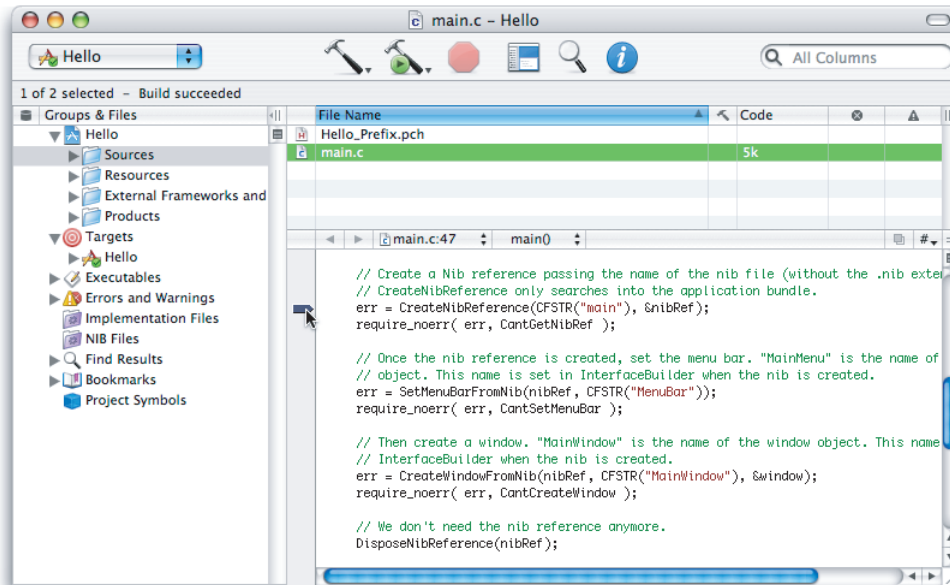
## Run-time Debugging

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 and see how it works.

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

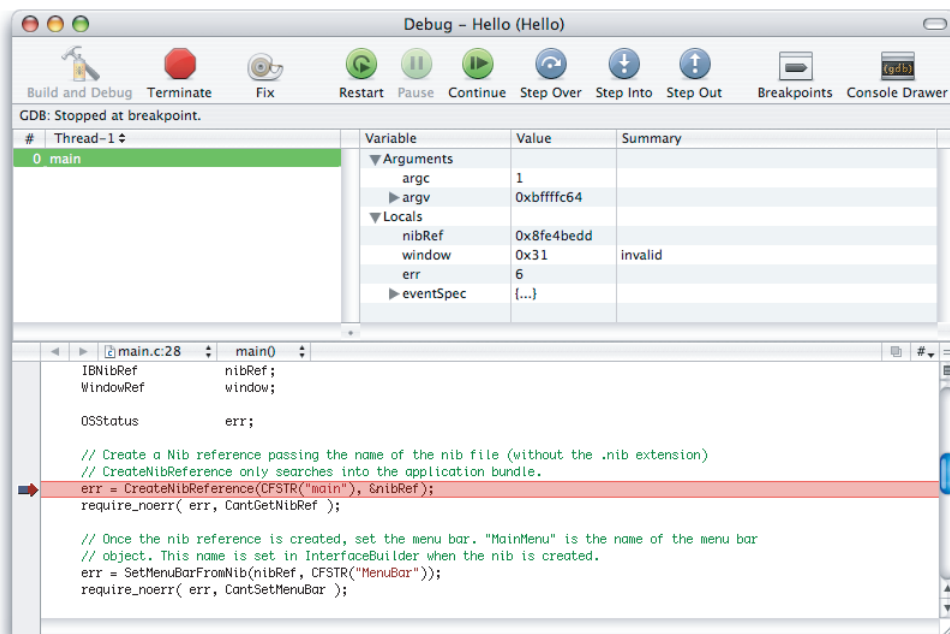
1. Open the file `main.c`. As before, you can open the file inside the project window or in a separate editor window.
2. Find the line that calls the function `CreateNibReference`.
3. Set a breakpoint by clicking in the column to the left of the function call, as shown in Figure 1-8.

Figure 1-8 Setting a breakpoint



4. Choose **Debug > Debug Executable (Command-Option-Y)**. Xcode opens the debugger window and runs the application in debug mode, pausing at the breakpoint you set (see Figure 1-9).

Figure 1-9 The Debug window



5. Using the **Step Over** command (Shift-Command-O), 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 Restart button to terminate and start a new debug session.

6. When you try to step over `RunApplicationEventLoop`, the program begins to run. Click the Terminate button to end the debug session.

## Summary

---

In this tutorial, you took a brief tour of the Xcode application.

The next tutorial shows how you can use the built-in indexing and searching features in Xcode to quickly find and display Apple technical documentation.



# Finding Technical Information

---

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

In this tutorial, you'll try out some of these features.

## Finding Technical Documentation

---

Apple publishes an extensive library of Mac OS X reference and conceptual documentation. This library is included in the Xcode developer tools package.

The built-in documentation window in Xcode provides a consistent, intuitive user interface and a wide range of search options for viewing the HTML-based content in this library. It's worth noting that Xcode uses the same HTML rendering engine used in Safari, Apple's Internet browser.

In this section, you will learn how to search for both reference and conceptual documentation.

## Searching For All Relevant Documentation

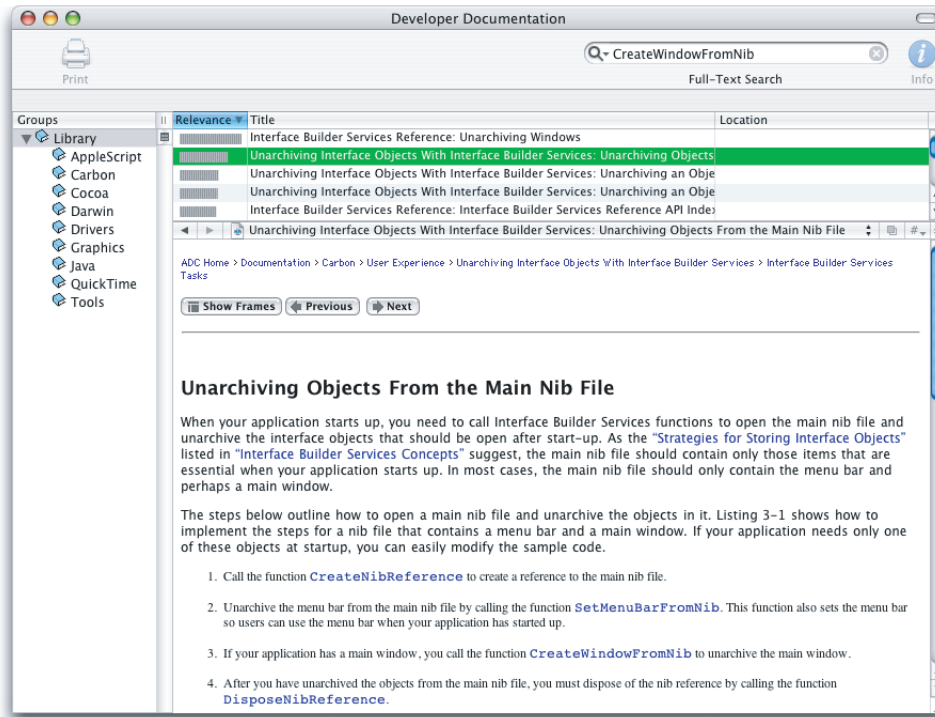
---

The Full-Text Search mode in the documentation window lets you search Apple's reference and conceptual documentation for articles relevant to a particular search string.

To search all technical documentation:

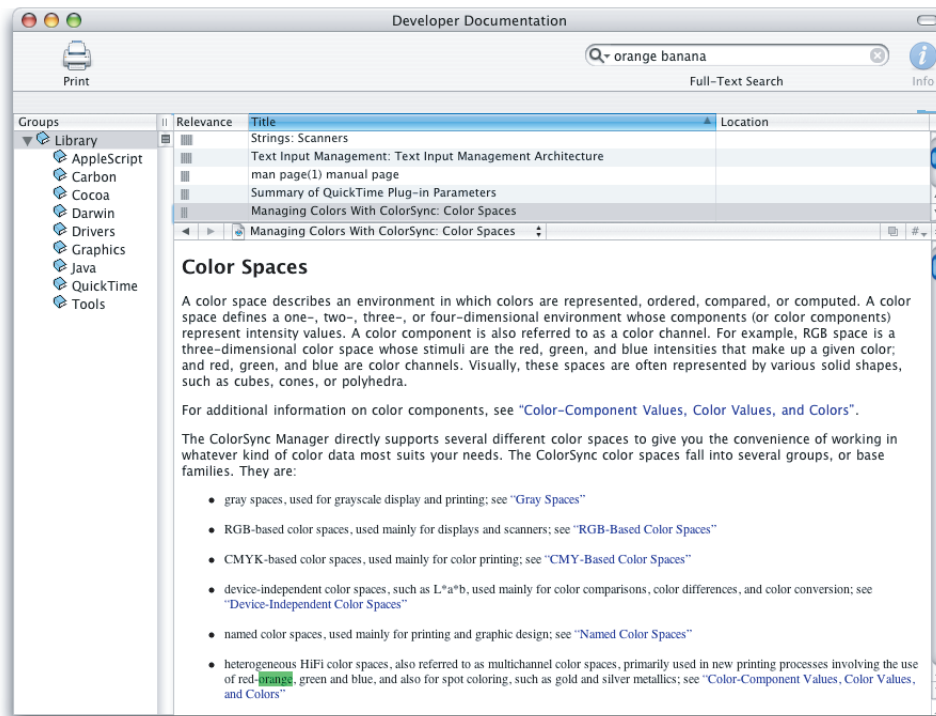
1. Launch Xcode and choose Help > Show Documentation Window. Xcode opens the window and displays the main documentation page.
2. From the pull-down menu in the Search field, choose Full-Text Search.
3. Type the word `CreateWindowFromNib` and press return. Xcode finds all documents that contain this word and lists them in order of relevance.
4. Select one of the items listed in the search results. Xcode displays this page of documentation in the viewing pane, as illustrated in Figure 2-1.

Figure 2-1 Using the full-text search option in Xcode



5. Now type orange banana into the Search field, and press return. Note that Xcode searches for each word separately and combines the search results into a single list, as illustrated in Figure 2-2.

Figure 2-2 Full-text search with two search terms



## Following Links

The documentation in [Figure 2-1](#) (page 22) and [Figure 2-2](#) (page 23) contains a number of hypertext links to other pages in the library. By convention, links are displayed as underlined 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 displays the linked page in an Internet browser window.

## Searching For API Documentation

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

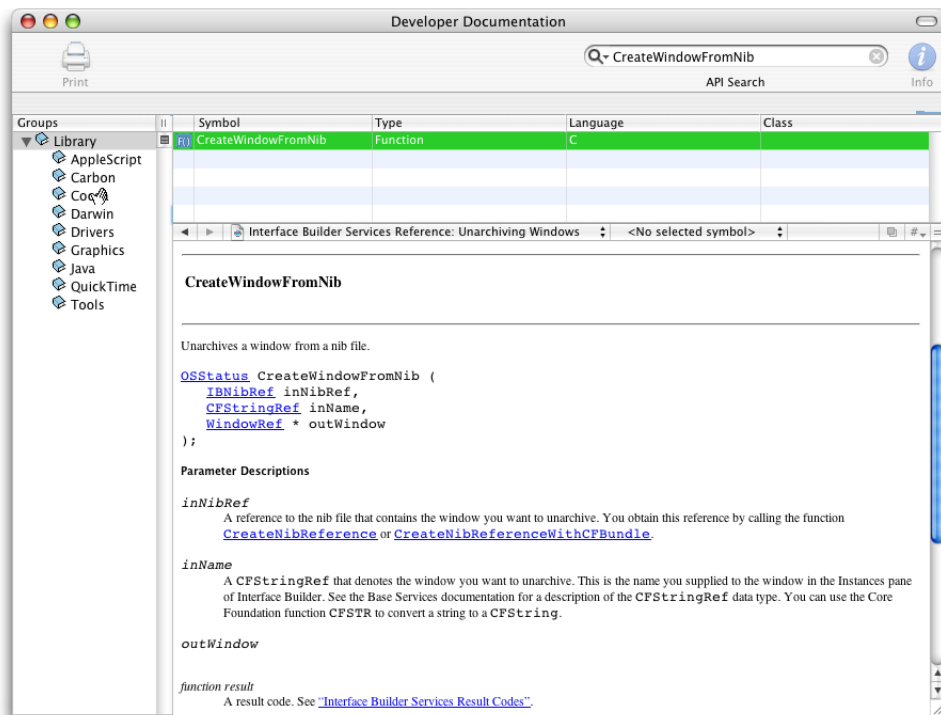
## Searching From the Documentation Window

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 open the documentation window.
2. In the Groups list, select Library or Carbon to define the scope of the search.
3. From the pull-down menu in the Search field, choose API-Reference Search.
4. Click in the Search field and begin typing the name of the function `CreateWindowFromNib`. 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 `CreateWindowFromNib`. As shown in Figure 2-3, Xcode finds and displays the API documentation for this function in the viewing pane.

**Figure 2-3** Using the API-reference search option in Xcode



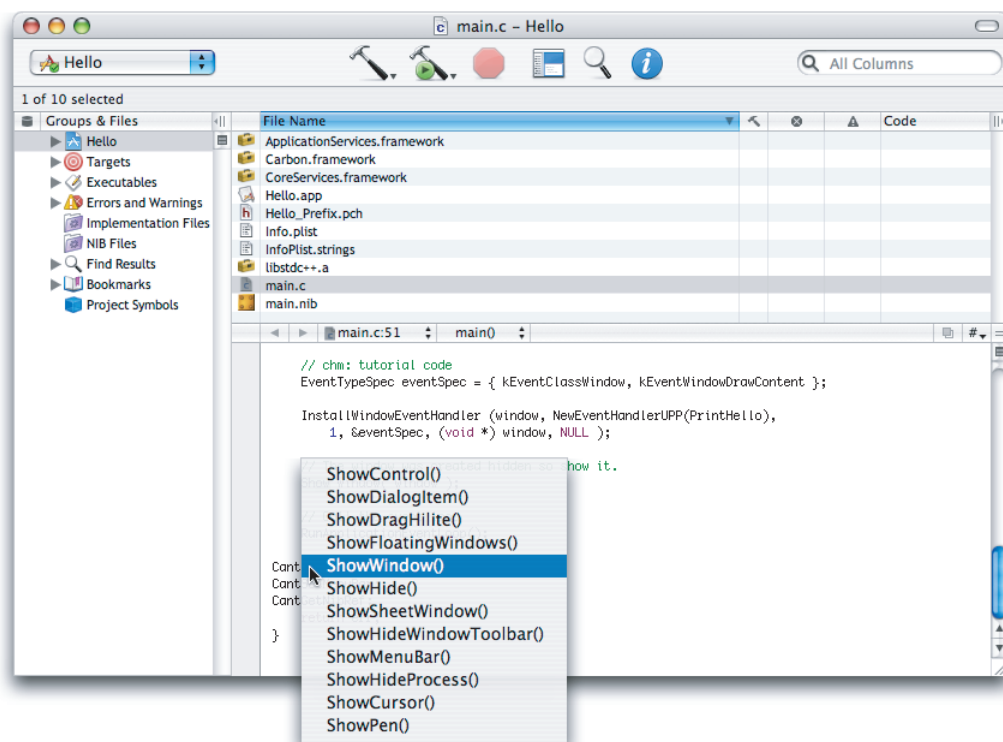
## Searching From a Source Editor Window

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



1. Open the Hello project window.
2. Open the source file `main.c` in an Xcode editor window.
3. Scroll down and find the function `ShowWindow`.
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.
5. Return to `main.c` and edit the name of the function to make it two words—that is, add a space between `Show` and `Window`.
6. Option-double-click the word `Show`. Now Xcode displays a contextual menu with a list of all the symbols that start with `Show`, as shown in Figure 2-4.

**Figure 2-4** Partial symbol search



7. Choose an item in the contextual menu, and verify that reference information for that symbol is displayed in the documentation window.

## 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

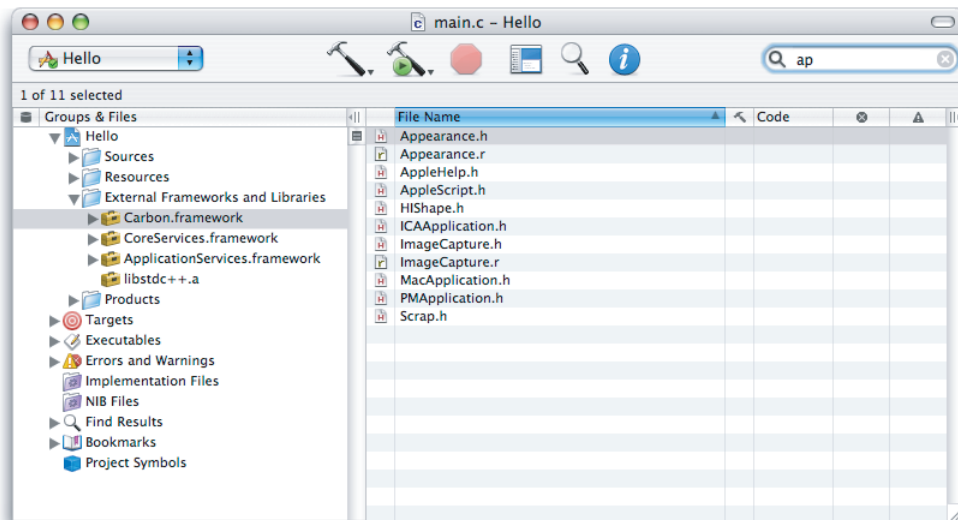
---

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 Carbon framework:

1. In the project window, use the disclosure triangles to open the project group. The folders inside are called source groups, and they contain references to actual files somewhere on your hard disk.
2. Open up the source group named External Frameworks and Libraries, and select Carbon.framework. In the detail view to the right, Xcode lists the header files contained in this framework.
3. Click in the Search field located in the upper right corner of the project window 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 2-5.

**Figure 2-5** Searching for headers in a framework group



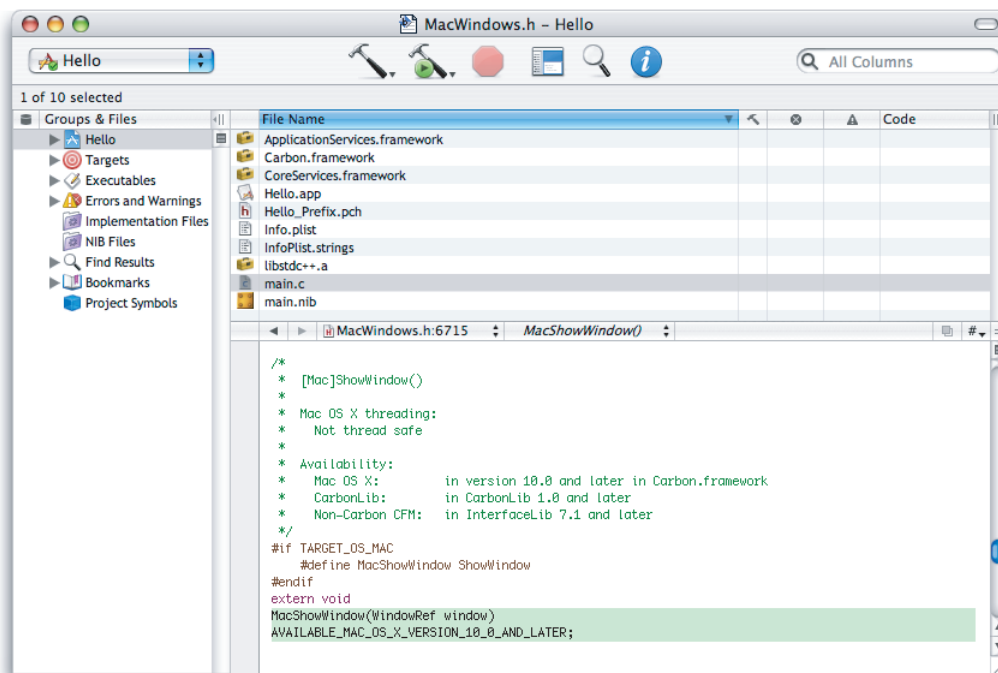
## Finding Symbol Declarations

In “[Searching From a Source Editor Window](#)” (page 24), you learned how to find the reference 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 `ShowWindow`:

1. From the Hello project window, open the source file `main.c`.
2. Find the line where the function `ShowWindow` 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 `MacWindows.h` and displays the declaration for `ShowWindow`, as shown in Figure 2-6.

**Figure 2-6** Header search results



## Summary

In this tutorial, you learned how to use Xcode to quickly locate technical information as you work on a project.

The next tutorial shows how you can use Interface Builder to design the user interface for a simple Carbon application.

**C H A P T E R 2**  
Finding Technical Information

# Designing a User Interface

---

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 Human Interface Guidelines
- Takes advantage of the newest technology in Carbon and Cocoa

In this tutorial, you will use Interface Builder to design the user interface for a Carbon application called Converter. After you're finished, you can incorporate your design into a working application.

Note: If you're planning to use Cocoa, you should also read *Developing Cocoa Objective-C Applications: A Tutorial*.

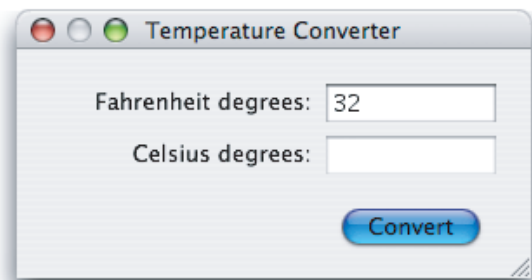
## Creating the Converter Interface

---

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

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

**Figure 3-1** The Converter user interface



To convert a temperature, you tab into the Fahrenheit field and enter a value. When you press return or click the Convert button, Converter displays the result in the Celsius field, which is read-only. To quit the application, you can either click the close button or type Command-Q.

## Getting Started

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

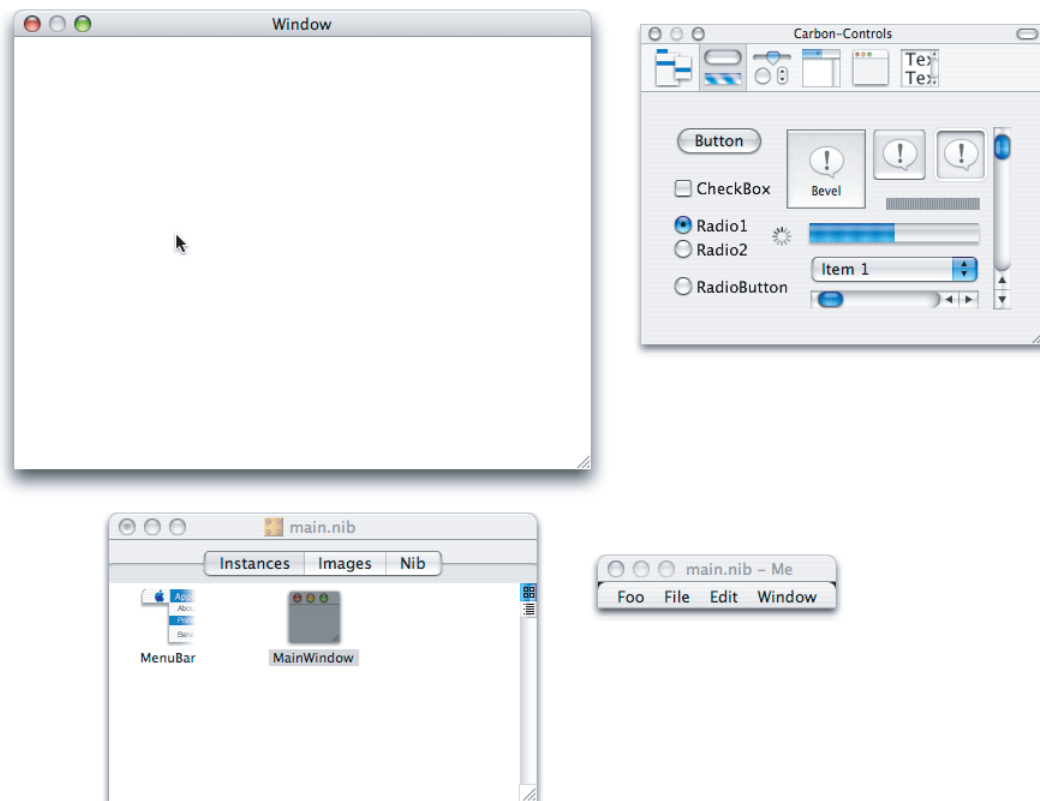
Xcode includes a template project that you can use to create a new Xcode project for the Converter application. This template includes a nib file that defines a default user interface.

To create the Converter project and open its nib file:

1. Use the procedure described in [“Creating an Xcode Project”](#) (page 9) to create a new project for a Carbon application. Give the project the name Converter.
2. In the Converter project window, locate and double-click the file called `main.nib`.

As shown in Figure 3-2, Interface Builder launches and presents a set of windows, including a design window (currently empty), a menu editor, a palette, and a window titled `main.nib` that represents the nib file itself.

**Figure 3-2** Editing windows in Interface Builder



## Setting the Window's Attributes

---

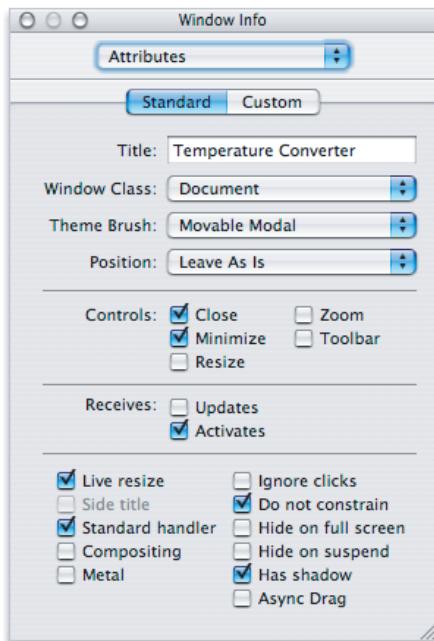
To set the attributes for the Converter window:

1. Click in the design window to select it.
2. Choose Tools > Show Info (Shift-Command-I).

This command opens an Interface Builder window called an Info window (see Figure 3-3). Sometimes referred to as an inspector, this window lets you view and change the attributes of any object in a nib file. There can be only one Info window open at a time, and it displays the attributes for the currently selected object.

3. From the pop-up menu at the top of the Info window, choose Attributes. The attributes for the Converter window are now visible.
4. Change the Converter window's title to "Temperature Converter", and press Return to make the change appear in the window.
5. From the Window Class menu, choose Document.
6. From the Theme Brush menu, choose Movable Modal. This theme gives the interior part of the window an Aqua look.
7. Uncheck Resize and Zoom—there's no need to support window zooming and resizing controls here.
8. Uncheck Compositing, since we're not using HView functionality in this tutorial. At this point, the window attributes should look like those in Figure 3-3.

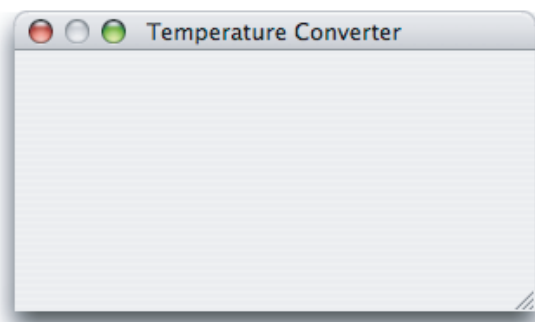
**Figure 3-3** Attributes of the Converter window



9. From the Info window's main pop-up menu, choose Size.
10. In the Content Rect group, set the window's width to 300 and 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 right, but they're a good first approximation. You will set the exact dimensions later, after adding the controls.

The design window should now resemble the window in Figure 3-4.

**Figure 3-4** Converter window before adding controls



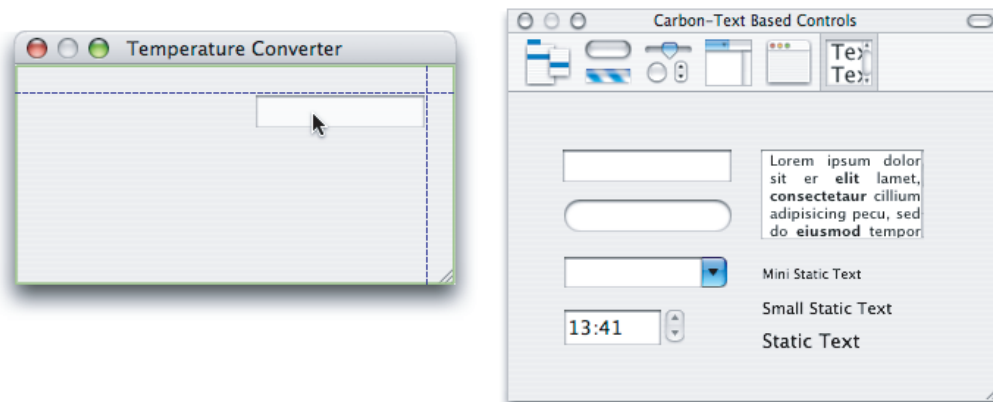
The Info window is used throughout this tutorial, so it makes sense to leave it open and available.

## Adding the Fahrenheit Control

To add the Fahrenheit control:

1. In the palette, select Carbon Text-Based Controls. You can see the names of the control groups by hovering the cursor over them in turn.
2. Drag an editable text field from the palette to the design window, as shown in Figure 3-5.

**Figure 3-5** Adding an editable text field





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.

3. Select the editable text field, and choose Attributes in the Info window pop-up menu.
4. In the Title field, type 32 and press Return to apply the value. This serves as the default Fahrenheit temperature.
5. Ensure the text field is still selected. From the Info window's main pop-up menu, choose Control.
6. All Carbon controls have a 4-byte application signature and a unique integer identifier. For the signature of this control, enter DEMO. For the ID, enter 128.
7. From the Info window pop-up menu, choose Size. Set the width to 92.
8. Back in the main window, adjust the position of the text field as needed.

## Adding the Celsius Control

---

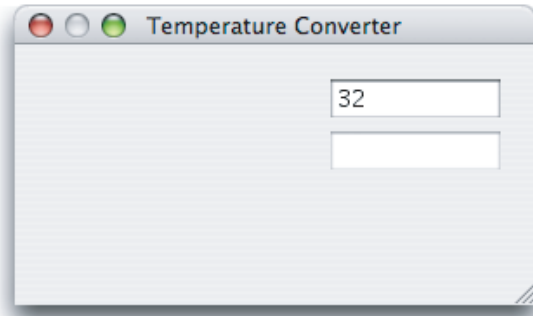
Converter needs a second control to display the Celsius temperature, the same size as the first.

To duplicate the Fahrenheit control:

1. In the main window, select the editable text field.
2. Choose Edit > Duplicate (Command-D). A new text field appears, slightly offset from the original.
3. Position the new text field under the original. Allow the guides to assist you by snapping the second field into place.
4. With the new field selected, go to the Info window and choose Attributes from the pop-up menu. Remove the title—a default Celsius temperature is not needed.
5. Choose Control, set the ID to 129, and uncheck the Enabled checkbox to make this a read-only control.

The design window should now resemble the window in Figure 3-6.

**Figure 3-6** Converter window after adding temperature controls



## Adding Static Text Labels

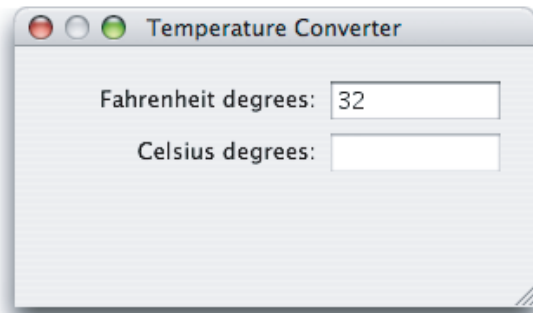
---

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

1. Drag a static text field from the Text-Based Controls palette to the design window. For now, you can place it in the empty area below the temperature controls.
2. With the static text field selected, go to the Info window and choose Attributes from the pop-up menu. Set the title to "Fahrenheit degrees:" and set the Justification attribute to Right.
3. Now choose Size from the pop-up menu, and set the width of this field to 150.
4. Back in the main window, position the static text field to the left of the first editable text field. Allow the guides to assist you by snapping it into place.
5. Make a duplicate static text field. Give it the title "Celsius degrees:", set the Justification attribute to Right, and position it to the left of the second editable text field.

The design window should now resemble the window in Figure 3-7.

**Figure 3-7** Converter window after adding static text



## Adding the Convert Button

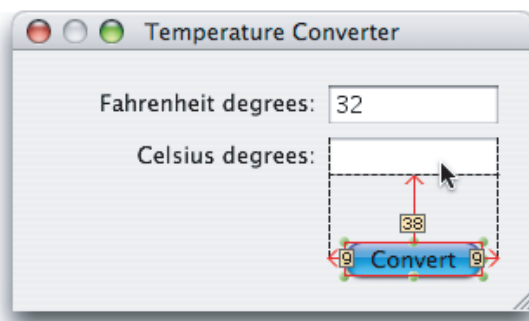
---

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

To add the Convert button:

1. Drag a button from the Carbon Controls palette to the lower-right portion of the window.
2. With the button selected, choose Attributes in the Info window pop-up menu and set the button's title to "Convert".
3. Set the Button Type attribute to Default. In a user interface with a default button, pressing the return key simulates a button click.
4. From the Info window pop-up menu, choose Control. Set the signature to `DEMO`, the ID to 130, and the command code to `Conv`. When the user clicks the Convert button or presses the return key, a Carbon event with this command code is sent to the application.
5. In the Button Info window, choose Size and set the button width to 80.
6. Return to the main window, and align the button under the temperature controls. First, drag the button downward until the Aqua guide appears, and release the mouse. With the button still selected, hold down the Option key. If you move the cursor around, Interface Builder will show you the distance from the button to the object that you've indicated with the cursor. With the Option key still down and the cursor over the Celsius EditText control, use the arrow keys to nudge the button horizontally to the exact center of the text fields (see Figure 3-8).

**Figure 3-8** Measuring distances in Interface Builder



## Aqua Layout and Object Alignment

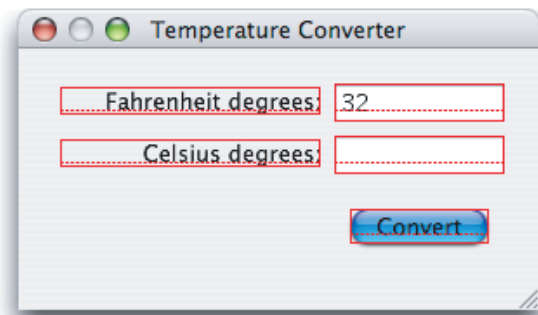
---

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

Because interface objects have shadows, they don't visually align correctly if you align the edges of the frames (as in Mac OS 9). For example, the Aqua guidelines say that a push button should be 20 pixels tall, but you actually need a frame of 32 pixels for both the button and its shadow. The layout rectangle is what you must align.

To view the layout rectangles in a design window, choose Show Layout Rectangles in the Layout menu (Command-L). Figure 3-9 shows the layout rectangles in the Converter interface.

**Figure 3-9** Layout rectangles in Interface Builder



Interface Builder gives you many ways to align objects in a window:

- Dragging objects with the mouse in conjunction with Aqua guides
- Pressing arrow keys (with the grid off, the selected objects move 1 pixel)
- Using a reference object to put selected objects in rows and columns
- Using the built-in alignment functions
- Specifying origin points in the Size display of the Info window
- Using horizontal and/or vertical guides

Look in the Alignment and Guides submenus of the Layout menu for various alignment commands and tools. You can also use the alignment tool (choose Alignment from the Tools menu). This tool provides a floating window with buttons that perform various types of alignment.

## Finishing the Window Layout

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

1. Select all of the controls in the main window either by dragging a rectangle around them or by typing Command-A.
2. Choose Layout > Group (Command-G) to group them together. Interface Builder uses a stippled rectangle to indicate the object group.
3. Drag the group (using any control as a handle) to the upper-left corner of the window. Use the Aqua guides to help you find the proper position relative to the edges of the window.

4. Choose Layout > Ungroup (Shift-Command-G).
5. If necessary, move the button into position again under the temperature controls.
6. Now resize the window, using the guides to give you the proper distance from the text fields (on the right) and the Convert button (on the bottom).

At this point, the main window should look like [Figure 3-1](#) (page 29).

## 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 All to save your work.
2. Choose File > Test Interface to run the simulator.
3. Try tabbing into the Fahrenheit field, entering a value, and cutting and pasting the value.
4. Make sure the Convert button is Aqua compliant (a default button with a pulsating, blue appearance).
5. Note that the screen position of the window in Interface Builder is used as the initial position for the window when Converter is launched.
6. When finished, choose Carbon Simulator > Quit (Command-Q).

## Implementing the Converter Application

---

Appendix A, “[Converter Source Code](#)” (page 45), contains the source code to implement the Converter application. Because this tutorial is not designed to teach you Carbon programming, an explanation of the code is not provided.

To add this source code to the project:

1. Open the Converter project, and open `main.c` in a separate editor window.
2. Choose File > Save a Copy As and save a backup copy of this file.
3. Replace the contents of this file with the source code in [Listing A-1](#) (page 45).
4. Save the file.
5. Build and run the application to make sure it works properly.

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

- Edit the main menu bar using Interface Builder. The default Carbon application menu 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 F.
- Add conversion from Celsius to Fahrenheit.

## Summary

---

In this tutorial, you used Interface Builder to design a user interface for a Carbon application, and you incorporated your design into a working Xcode project.

The last tutorial introduces a powerful debugging feature in Xcode called Fix and Continue.

# Using Fix and Continue

Up to now, you haven't seen any of the advanced features in the Xcode application. However, a new feature called Fix and Continue is so interesting, it's worth an early look.

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 if it takes a lot of time to reach your application's current state of execution in the debugger. Rather than recompile your project and restart your debugging session, you can make minor changes to your code, fix your executable image, and see the immediate results of your changes.

The following short tutorial uses a sample project named Sketch, 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.

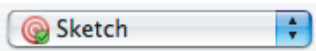
You don't need to be familiar with Objective-C to work through this tutorial.

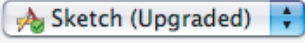
## Configuring the Project

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 project folder onto the Xcode application, which is located in `/Developer/Applications`. Notice that Xcode finds and opens the project file inside this folder.
3. To use the Fix command, you need to build Sketch using the native build system in Xcode. In the active target pop-up menu, if the target's icon looks like the icon in the bottom row in Table 4-1, the target is already native and you can skip to step 6.

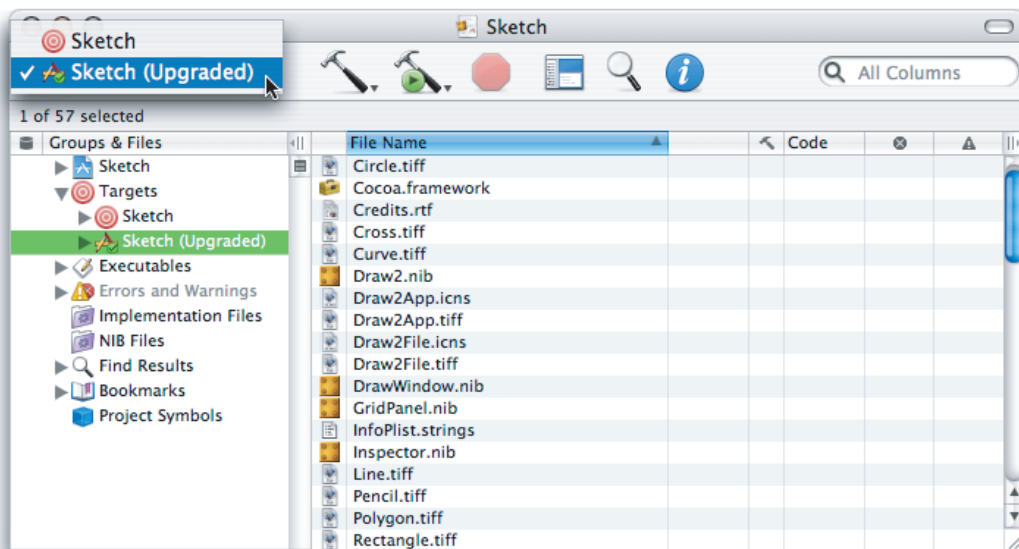
**Table 4-1** Types of build targets

Target type	Build system	Appearance in the Target pop-up menu
Project Builder	jam	

Target type	Build system	Appearance in the Target pop-up menu
Xcode	native	

- In Groups & Files, open the Targets group and select the Sketch target.
- Choose Project > Upgrade Target To Native, and click Upgrade.
- From the active target pop-up menu, choose the native target. This step is illustrated in Figure 4-1.

**Figure 4-1** Choosing the native build target



- Select Sketch in the Groups & Files list, and choose Project > Get Info (Command-I) to open the Info window.
- Click Styles. (In Xcode, the settings in the active build style override the same settings in the active target.)
- From the Active Build Style pop-up menu, choose Development. Confirm that Zero link is checked, and that Optimization level is None.
- From the Build menu, choose Build (Command-B) to build the target.

## Fixing the Running Application

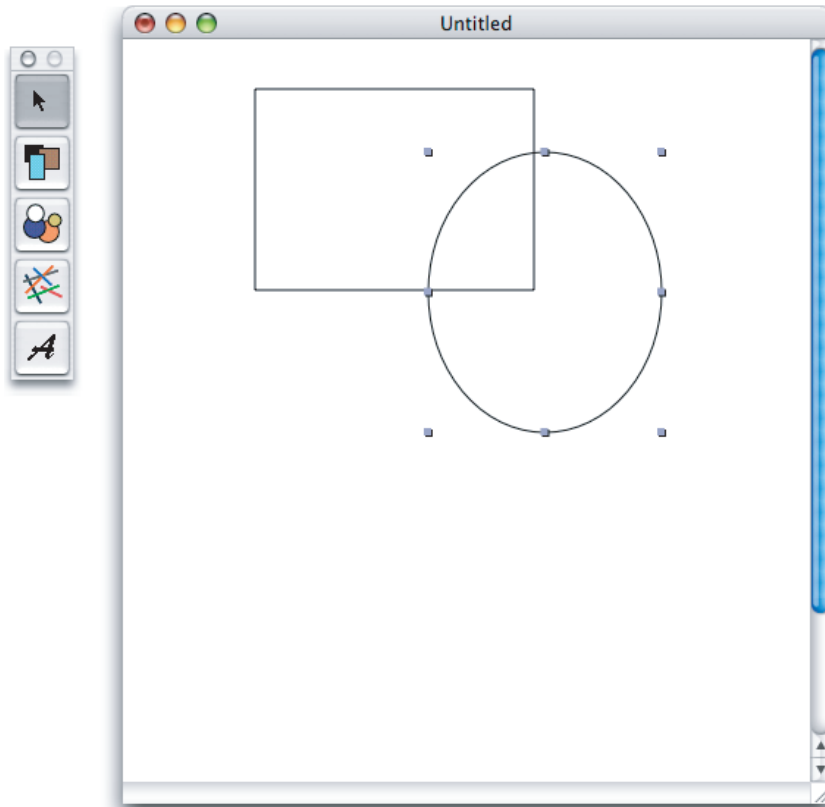
---

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



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

**Figure 4-2** The default drawing behavior of Sketch



- Return to the project window, which includes all the source files in the current target. Open the source file named `SKTGraphic.m`.
- In the Editor window, you'll find the `-[SKTGraphic init]` function. It looks like this:

```
- (id)init {
    self = [super init];
    if (self) {
        _document = nil;
        [self setBounds:NSMakeRect(0.0, 0.0, 1.0, 1.0)];
        [self setFillColor:[NSColor whiteColor]];
        [self setDrawsFill:NO];
        [self setStrokeColor:[NSColor blackColor]];
        [self setDrawsStroke:YES];
        [self setStrokeLineWidth:1.0];
        _origBounds = NSZeroRect;
        _gFlags.manipulatingBounds = NO;
    }
    return self;
}
```

- Change some of the values.

For example, change:

```
[self setFillColor:[NSColor whiteColor]];
[self setDrawsFill:NO];
[self setStrokeColor:[NSColor blackColor]];
[self setDrawsStroke:YES];
[self setStrokeLineWidth:1.0];
```

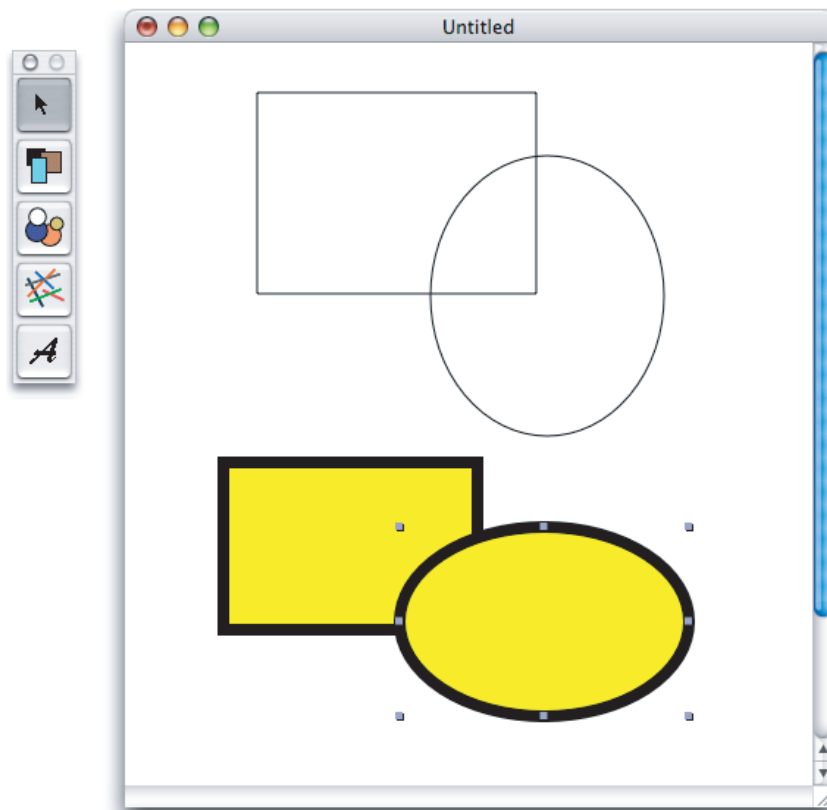
to:

```
[self setFillColor:[NSColor yellowColor]];
[self setDrawsFill:YES];
[self setStrokeColor:[NSColor blackColor]];
[self setDrawsStroke:YES];
[self setStrokeLineWidth:8.0];
```

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

You should see a dramatic change in the appearance of the objects you draw, as illustrated in Figure 4-3.

**Figure 4-3** The new drawing behavior of Sketch



## Summary

---

Fix and Continue is a powerful way to make small changes to a source file without restarting your debugging session. Use of the Fix command is subject to certain requirements and restrictions, which are fully documented in the programming topic Fix and Continue.

Congratulations, you have finished this quick tour of Xcode!



# Converter Source Code

Listing A-1 contains the source code for the Converter application described in [“Designing a User Interface”](#) (page 29).

## Listing A-1 Converter source code

```
#include <Carbon/Carbon.h>

static OSStatus Convert (WindowRef window)
{
    const ControlID kFahrenheit = { 'DEMO', 128 };
    const ControlID kCelsius = { 'DEMO', 129 };

    ControlRef fahrenheit, celsius;
    CFStringRef fahrenheit_text, celsius_text;
    double fahrenheit_value, celsius_value;
    OSStatus err = noErr;

    err = GetControlByID (window, &kFahrenheit, &fahrenheit);
    require_noerr_quiet (err, CantGetControl);

    err = GetControlByID (window, &kCelsius, &celsius);
    require_noerr_quiet (err, CantGetControl);

    err = GetControlData (fahrenheit, 0, kControlEditTextCFStringTag,
        sizeof(CFStringRef), &fahrenheit_text, NULL);
    require_noerr (err, CantGetControlData);

    fahrenheit_value = CFStringGetDoubleValue (fahrenheit_text);
    celsius_value = (fahrenheit_value - 32.0) * 5.0 / 9.0;

    celsius_text = CFStringCreateWithFormat (
        NULL, NULL, CFSTR("%g"), celsius_value);

    err = SetControlData (celsius, 0, kControlEditTextCFStringTag,
        sizeof(CFStringRef), &celsius_text);

    CFRelease (celsius_text);
    require_noerr (err, CantSetControlData);

    DrawOneControl (celsius);

    CantSetControlData:
    CantGetControlData:
}
```

## Converter Source Code

```

CantGetControl:
    return err;
}

static OSStatus MainWindowCommandHandler (
    EventHandlerCallRef nextHandler, EventRef event, void *userData)
{
    HICommand command;
    WindowRef window = (WindowRef) userData;
    OSStatus err = noErr;

    err = GetEventParameter (event, kEventParamDirectObject,
        typeHICommand, NULL, sizeof(HICommand), NULL, &command);
    require_noerr (err, CantGetParameter);

    switch (command.commandID)
    {
        case 'Conv':
            Convert (window);
            break;
        default:
            err = eventNotHandledErr;
            break;
    }
}

CantGetParameter:
    return err;
}

static OSStatus MainWindowEventHandler (
    EventHandlerCallRef nextHandler, EventRef event, void *userData)
{
    OSStatus err = noErr;

    switch (GetEventKind (event))
    {
        case kEventWindowClosed:
            QuitApplicationEventLoop();
            break;
        default:
            err = eventNotHandledErr;
            break;
    }

    return err;
}

static OSStatus CreateMainWindow()
{
    IBNibRef nibRef;
    WindowRef window;
    OSStatus err;

    const EventTypeSpec commands[] = {
        { kEventClassCommand, kEventCommandProcess }
    };
};

```

Converter Source Code

```

const EventTypeSpec events[] = {
    { kEventClassWindow, kEventWindowClosed }
};

err = CreateNibReference(CFSTR("main"), &nibRef);
require_noerr( err, CantGetNibRef );

err = SetMenuBarFromNib(nibRef, CFSTR("MenuBar"));
require_noerr( err, CantSetMenuBar );

err = CreateWindowFromNib(nibRef, CFSTR("MainWindow"), &window);
require_noerr( err, CantCreateWindow );

DisposeNibReference(nibRef);

err = InstallWindowEventHandler (window,
    NewEventHandlerUPP (MainWindowEventHandler),
    GetEventTypeCount(events), events,
    window, NULL);

require_noerr (err, CantInstallHandler);

err = InstallWindowEventHandler (window,
    NewEventHandlerUPP (MainWindowCommandHandler),
    GetEventTypeCount(commands), commands,
    window, NULL);

require_noerr (err, CantInstallHandler);

ShowWindow( window );

CantInstallHandler:
CantCreateWindow:
CantSetMenuBar:
CantGetNibRef:

    return err;
}

int main(int argc, char* argv[])
{
    OSStatus err = CreateMainWindow();
    RunApplicationEventLoop();
    return err;
}

```





# Revision History

---

The table below lists the revisions to *A Quick Tour of Xcode*.

Date	Notes
2003-08-28	First public version of this document.
2003-06-20	Released as a preliminary document for WWDC 2003.

R E V I S I O N   H I S T O R Y

Revision History