CHAPTER 8

Mac OS X and Code

Up to this point, we've spent quite a bit of time buried in the weeds with user tools. To a large extent, we've ignored the fact that, aside from providing beautiful and functional user applications, Mac OS X is also a powerful development platform. Not only is Mac OS X loaded with the native Apple developer tools, including Xcode, but it's also very extensible using open source tools, such as Python, Perl, and PHP. A developer can create Cocoa applications native to Mac OS X, or use the Apple IDE to create tools that leverage the open source underpinnings of Mac OS X. It's also possible to add frameworks to the system that further extend the connections between these open source technologies and the native Apple development tools.

Mac OS X includes all the scripting tools you would expect in a Linux distribution. With Python, PHP, Perl, and Ruby, Mac OS X provides a powerful scripting platform, perfect for when full-blown object-oriented development is overkill. It's easy to create scripts that "do one thing and do it well." Apple also addresses this need in Mac OS X with AppleScript, a clean and easy-to-learn native scripting language.

Finally, as a developer, you understand the importance of source code and revision control. Mac OS X provides these tools as well. If you're a lone developer, it's important to understand the code control options that are available for your local machine or local network. If you work in a development house with several developers, you'll be interested in how Mac OS X can provide centralized code and revision control options, accessible on the network to all your developers. Again, the options for code control include native Mac OS X applications as well as tried-and-true open source alternatives, GUI options, and command-line tools. The flexibility you've seen in the user tools applies equally well to the developer tools in Mac OS X.

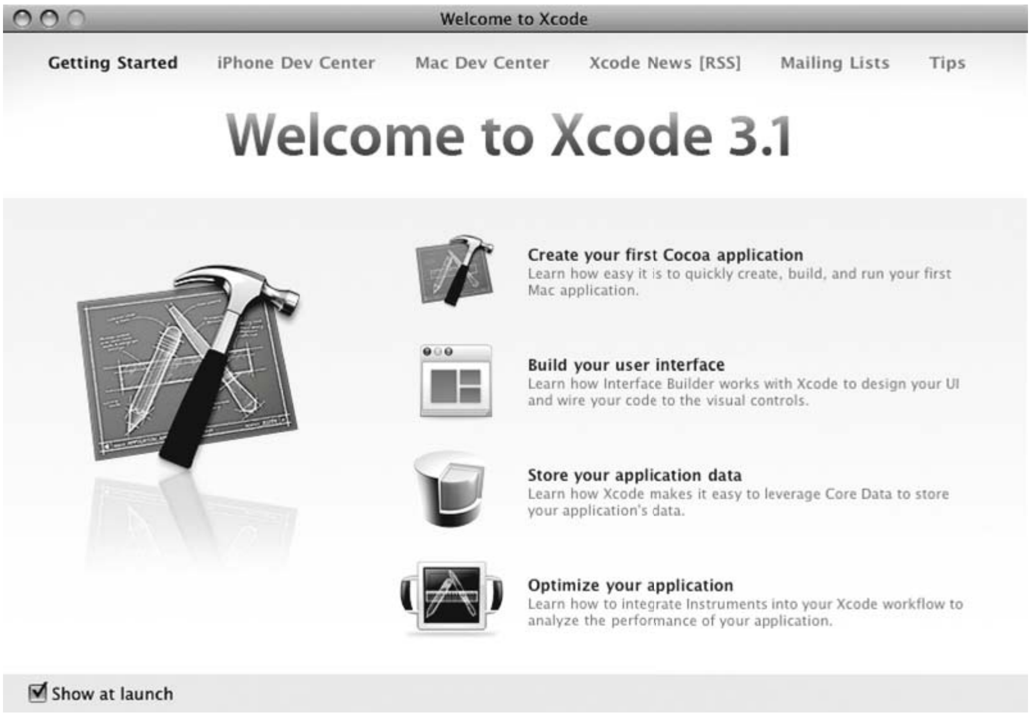
In this chapter, we'll cover development with Xcode, scripting, and code maintenance and revision control. We'll begin with a look at the Xcode IDE.

Using Xcode

As discussed in Chapter 4, Xcode is the Apple development environment provided on the Mac OS X installation DVD. Xcode has everything necessary to develop and debug applications for the Mac. Xcode is a complete IDE, including a text editor, build system, debugger, and compiler. It's the central point for development of nearly all the applications on the Mac, and the very same IDE used by developers within Apple.

Instructions for installing Xcode from the Mac OS X installation DVD are provided in Chapter 4. Once installed, the Xcode tools can be found, by default, on the `boot` volume. These tools are installed in the `Applications` subdirectory of the `Developer` folder on that volume.

As shown in Figure 8-1, the main Xcode window offers all the Xcode options. You can jump right into the IDE by selecting "Create your first Cocoa application." Alternatively, you can choose the other options to work with the Interface Builder, build a new database for your application data with Core Data, or use Instruments to optimize your application.



Create your first Cocoa application
Learn how easy it is to quickly create, build, and run your first Mac application.

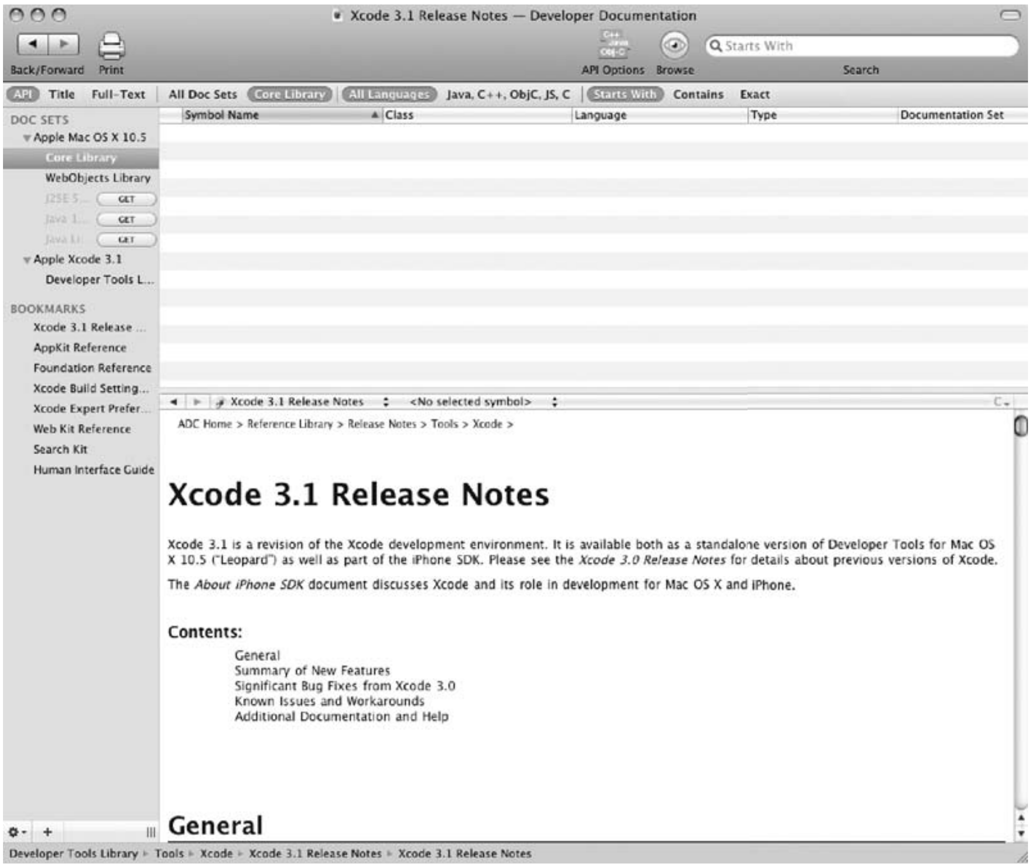
Build your user interface
Learn how Interface Builder works with Xcode to design your UI and wire your code to the visual controls.

Store your application data
Learn how Xcode makes it easy to leverage Core Data to store your application's data.

Optimize your application
Learn how to integrate Instruments into your Xcode workflow to analyze the performance of your application.

Figure 8-1. *Launching the Xcode IDE in Mac OS X*

At the top of the main Xcode window, you'll also find links to internal documentation on a range of topics. These include the iPhone Dev Center, the Mac Dev Center, the latest news on Xcode, informational mailing lists on development and Xcode-related topics, and tips for using the Xcode tools. These links point to documentation libraries built into the Xcode tool. In order to preserve storage space, many of these libraries are initially populated with minimal information locally, with the complete documentation set being available online. Additionally, some documentation sets (notably those about Java) are not available at all locally. The complete documentation libraries can be downloaded and installed locally, if needed, by clicking the Subscribe button in the left pane of the window for the documentation set you would like to install. Additionally, the documentation tool contains bookmarks, also listed in the left pane, as shown in Figure 8-2. These open the documentation in the lower pane of the main screen in the documentation library.



Xcode 3.1 Release Notes

Xcode 3.1 is a revision of the Xcode development environment. It is available both as a standalone version of Developer Tools for Mac OS X 10.5 ("Leopard") as well as part of the iPhone SDK. Please see the *Xcode 3.0 Release Notes* for details about previous versions of Xcode.

The *About iPhone SDK* document discusses Xcode and its role in development for Mac OS X and iPhone.

Contents:

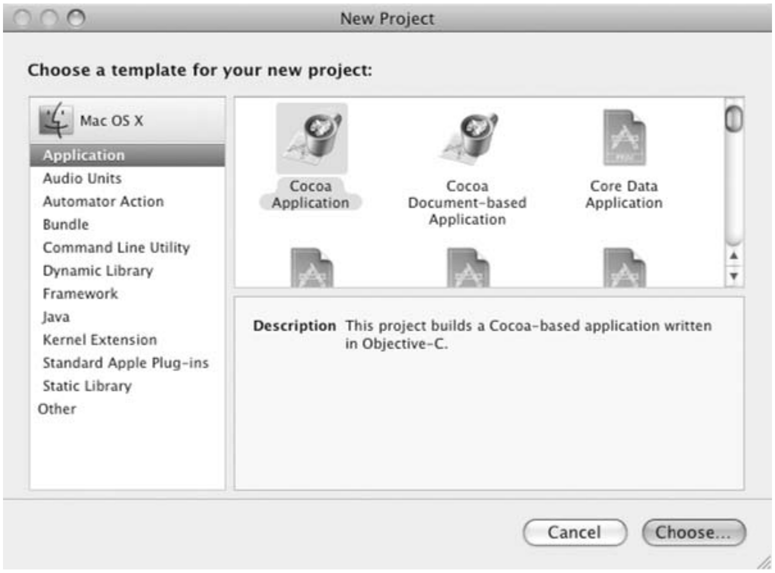
General
Summary of New Features
Significant Bug Fixes from Xcode 3.0
Known Issues and Workarounds
Additional Documentation and Help

General

Figure 8-2. *The online documentation libraries for the Xcode tools*

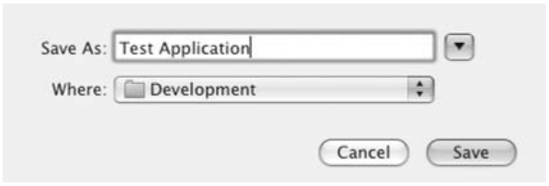
Creating an Application with Xcode

When starting a new project, Xcode provides a number of templates. As shown in Figure 8-3, these are grouped into several project types, including applications, Automator actions, kernel extensions, and several others. These templates provide the basic files for your projects. Select a template category, and then select a specific template for your project. Xcode prompts you for the location where all project files will be stored, as shown in Figure 8-4.



Description This project builds a Cocoa-based application written in Objective-C.

Figure 8-3. *Choosing a template for a new Xcode project*



Save As: Test Application

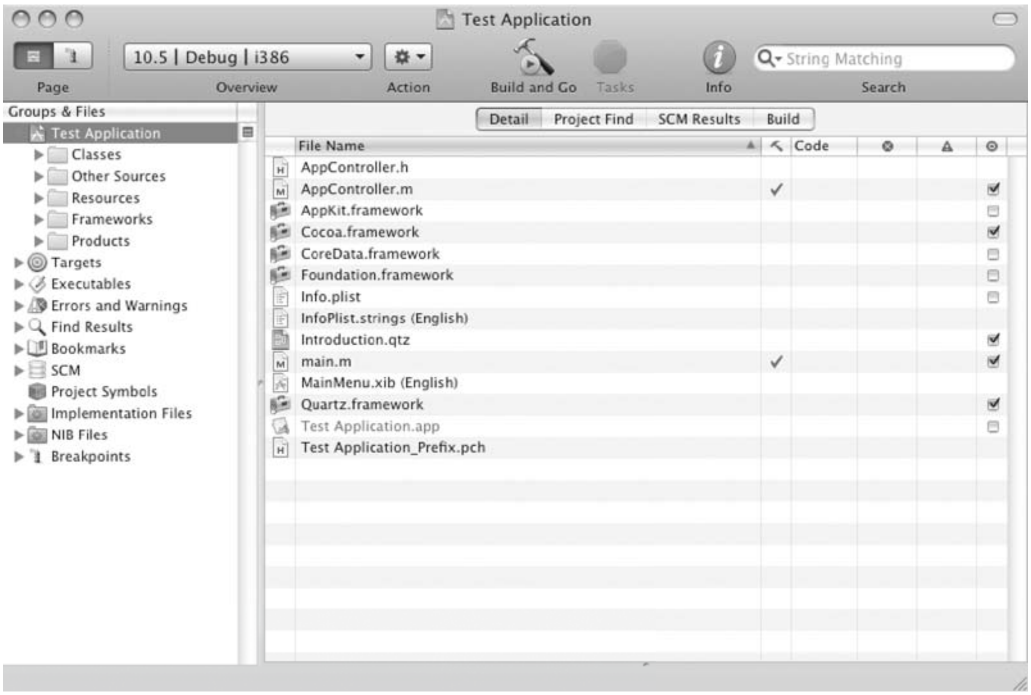
Where: Development

Figure 8-4. *Saving a new Xcode project*

As shown in Figure 8-5, the Xcode tool creates a basic library of files for the new application. These include header files, app files, necessary frameworks, and `plist` files, as well as several others. These are organized in the project within folders in the left pane of the Xcode window. The folders include classes, other sources, resources, frameworks, and products. Double-clicking a file name in the Xcode tool will open the file in the built-in text editor.

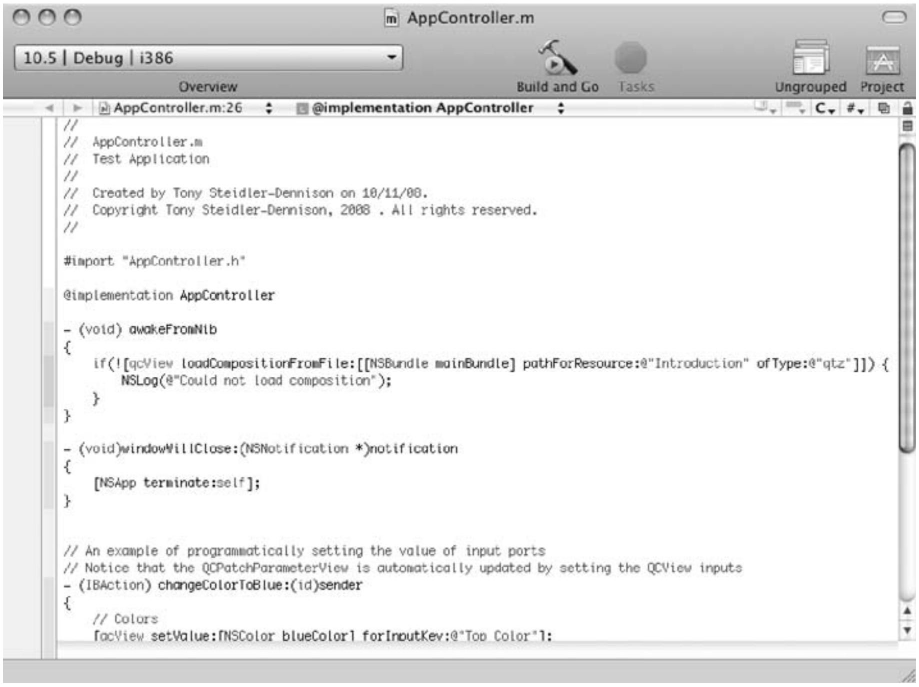
Xcode creates files with basic information in the header, including the code author and copyright information. This is based on information in the user's entry in the Address Book application. The files also contain the basic code necessary to create the specific file for the application, again, based on the chosen application type.

The Xcode text editor utilizes full syntax highlighting, in addition to inserting comments for the developer, specific to the project type. In the example shown in Figure 8-6, the comments provide the developer with information about the creation of a function for setting the value of input ports. This type of commenting is provided throughout all new project files.



File Name		Code			
AppController.h					
AppController.m	✓				☑
AppKit.framework					☐
Cocoa.framework					☑
CoreData.framework					☐
Foundation.framework					☐
Info.plist					☐
InfoPlist.strings (English)					
Introduction.qtz					☑
main.m	✓				☑
MainMenu.xib (English)					
Quartz.framework					☑
Test Application.app					☐
Test Application_Prefix.pch					

Figure 8-5. *The file library created by Xcode for a new project*



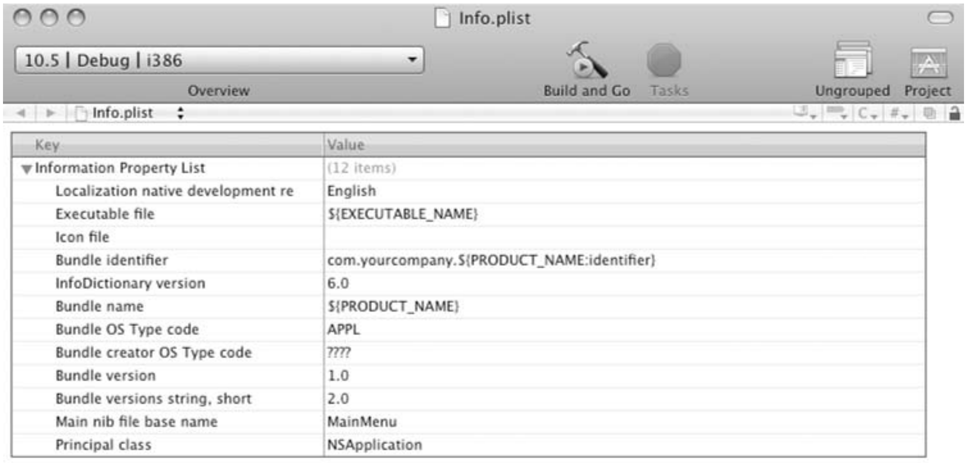
```
//
//  AppController.m
//  Test Application
//
//  Created by Tony Steidler-Dennison on 10/11/08.
//  Copyright Tony Steidler-Dennison, 2008 . All rights reserved.
//

#import "AppController.h"

@implementation AppController

- (void) awakeFromNib
{
    if(![qcView loadCompositionFromFile:[[NSBundle mainBundle] pathForResource:@"Introduction" ofType:@"qtz"]]) {
        NSLog(@"Could not load composition");
    }
}

- (void)windowWillClose:(NSNotification *)notification
{
    [NSApp terminate:self];
}


// An example of programmatically setting the value of input ports
// Notice that the QCPatchParameterView is automatically updated by setting the QCView inputs
- (IBAction) changeColorToBlue:(id)sender
{
    // Colors
    [qcView setValue:[NSColor blueColor] forInputKey:@"Top Color"];
```

Figure 8-6. *The AppController.m file open for editing in the built-in text editor*

Applications in Mac OS X require a property list, or `plist` file. These simple XML files describe the application and user settings required for the operation of the application. The `plist` files are one of the many direct descendants of the NeXTStep operating system upon which Mac OS X is built.

As each Mac OS X application requires a `plist` file, it's only appropriate that the Xcode tool would provide the means by which to create these files. As a recent addition to Xcode, the `plist` editor is included in Xcode 3.1 and later. To launch it, double-click the `Info.plist` object in the project's `Resources` folder. A default `plist` file is shown in Figure 8-7.

■ **Note** Though Xcode includes a `plist` editor, it's not actually necessary in order to create or modify `plist` files. As XML files, the `plist` files can be created and edited with any text editor.

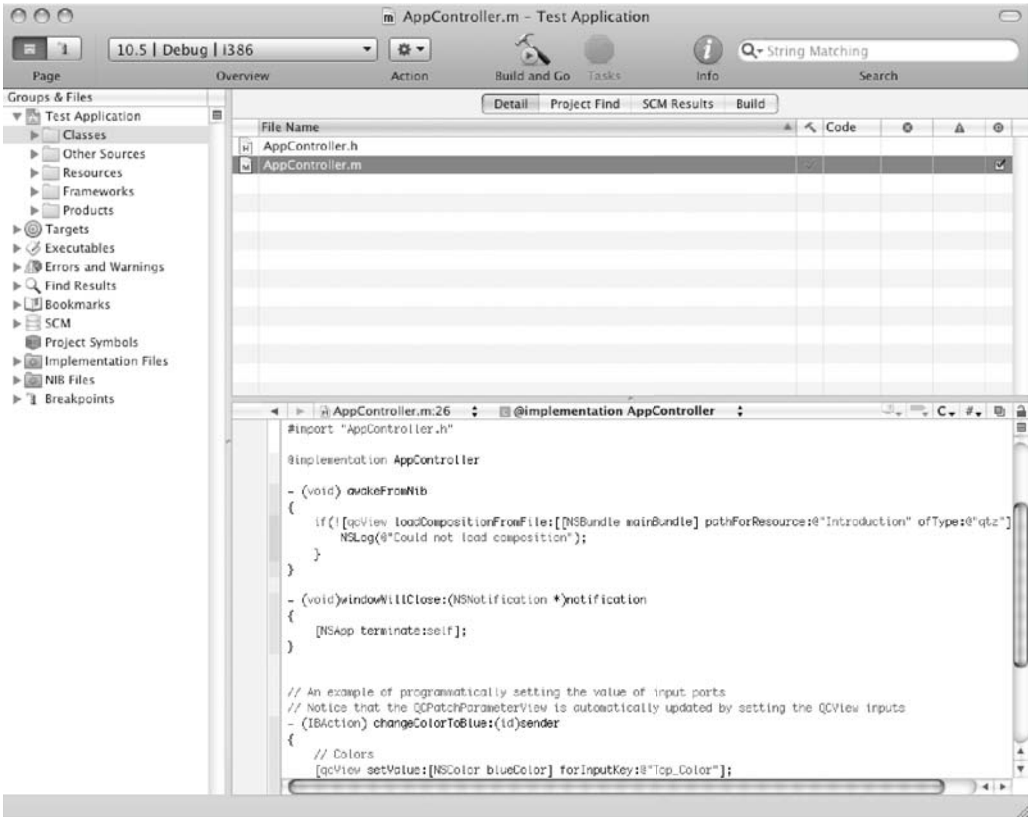


Key	Value
▼Information Property List	(12 items)
Localization native development re	English
Executable file	${EXECUTABLE_NAME}
Icon file	
Bundle identifier	com.yourcompany.${PRODUCT_NAME:identifier}
InfoDictionary version	6.0
Bundle name	${PRODUCT_NAME}
Bundle OS Type code	APPL
Bundle creator OS Type code	????
Bundle version	1.0
Bundle versions string, short	2.0
Main nib file base name	MainMenu
Principal class	NSApplication

Figure 8-7. *Creating a plist file for the new application*

Working in the Main Xcode Window

Rather than launching a single file in the built-in Xcode editor by double-clicking it, you can configure the tool to provide access to all files in a single window. You can also configure external editors in the Xcode preferences. Figure 8-8 shows this configuration, opened for the `AppController.m` file. In this configuration, the editor opens in the lower pane of the main window.



```
#import "AppController.h"

@implementation AppController

- (void) awakeFromNib
{
    if(![qcView loadCompositionFromFile:[[NSBundle mainBundle] pathForResource:@"Introduction" ofType:@"qtz"]
        NSLog(@"Could not load composition");
    }
}

- (void)windowWillClose:(NSNotification *)notification
{
    [NSApp terminate:self];
}


// An example of programmatically setting the value of input ports
// Notice that the QCPatchParameterView is automatically updated by setting the QCView inputs
- (IBAction) changeColorToBlue:(id)sender
{
    // Colors
    [qcView setValue:[NSColor blueColor] forInputKey:@"Top_Color"];
```

Figure 8-8. *An expanded view of the Xcode interface*

At the top left of the Xcode IDE window are options for configuring both the target environment and the window view, as shown in Figure 8-9. Your choices for window view, selected by clicking one of the two icons at the far left, are the main window or the debugging window (the bug spray icon). To designate the target environment for the application, select it from the drop-down menu in the main interface. In the example, the options for target are Mac OS X 10.4 or Mac OS X 10.5 (Target Setting).

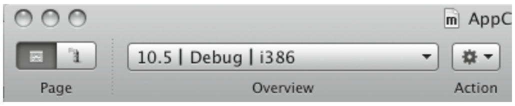


Figure 8-9. *The Xcode IDE settings*

Note Xcode settings include Project settings and Target settings. Without the parenthetical indication of which settings you're modifying, such as "(Target Setting)," it's possible to confuse the settings type. Changing these settings may, in fact, give unexpected results. In other words, it's important to pay close attention to the settings type when modifying the Xcode settings.

In addition to selecting the window view and the target environment for the application, Xcode also provides several actions that are easily accessible within the main interface. Click the Action drop-down arrow to see the list shown in Figure 8-10. You can select from the following types of actions:

- Open and add actions, such as adding a new file, opening the current file in the Finder, or opening the current file as a new file
- Actions specific to the individual file, such as viewing the file information (as provided, in part, by the existing `plist` file), renaming, touching, untouching, and deleting the file
- Actions to compile, preprocess, and show the assembly code for the file

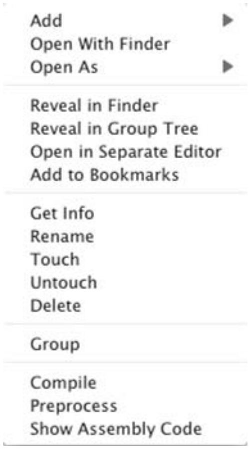


Figure 8-10. *Actions available in the Xcode interface*

All in all, the most common file actions are provided from within the main Xcode window.

As shown in Figure 8-11, Xcode provides other developer options within the main window, including the ability to build an executable binary with a single button click. When the Xcode IDE is actively building a binary or is engaged in some other task that temporarily excludes user interaction, the Tasks indicator in the main window bar will become active, painting the stop sign a bright red.

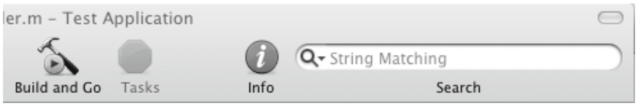
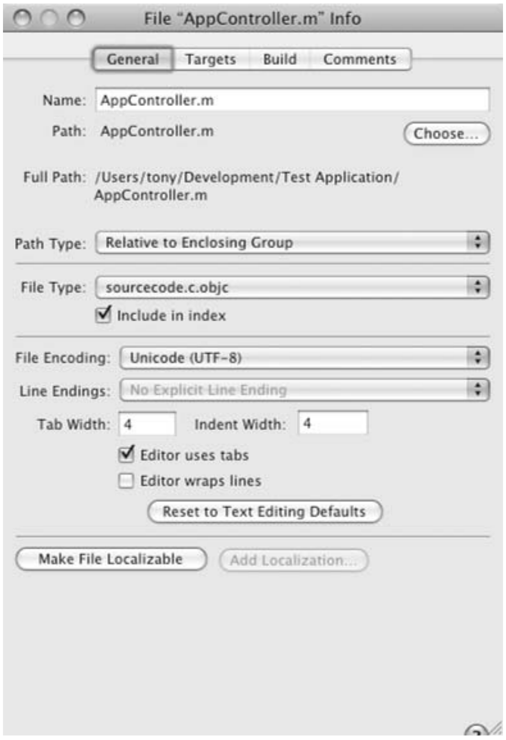


Figure 8-11. *Additional file options available within the Xcode main window*

In addition to the ability to build and compile from the main window, Xcode also provides a tool to view and edit all the available information about the current file within the main interface. The Info button at the top of the main Xcode window reveals another window with this information, as shown in Figure 8-12. This window includes several tabs of information, including general information, targets, additional compiler flags for the chosen application target, and file-specific comments.


File "AppController.m" Info

General | Targets | Build | Comments

Name: AppController.m
Path: AppController.m
Full Path: /Users/tony/Development/Test Application/AppController.m
Path Type: Relative to Enclosing Group
File Type: sourcecode.c.objc
☑ Include in index
File Encoding: Unicode (UTF-8)
Line Endings: No Explicit Line Ending
Tab Width: 4 Indent Width: 4
☑ Editor uses tabs
☐ Editor wraps lines

Figure 8-12. *Information about the current file available from the main Xcode interface window*

Finally, the main Xcode window includes a search interface. This provides developers with the ability to perform string searches within the selected file.

Debugging with Xcode

An essential part of creating applications is, of course, the ability to debug those programs you've created. Stepping through code execution one line at a time makes it possible to find coding errors.

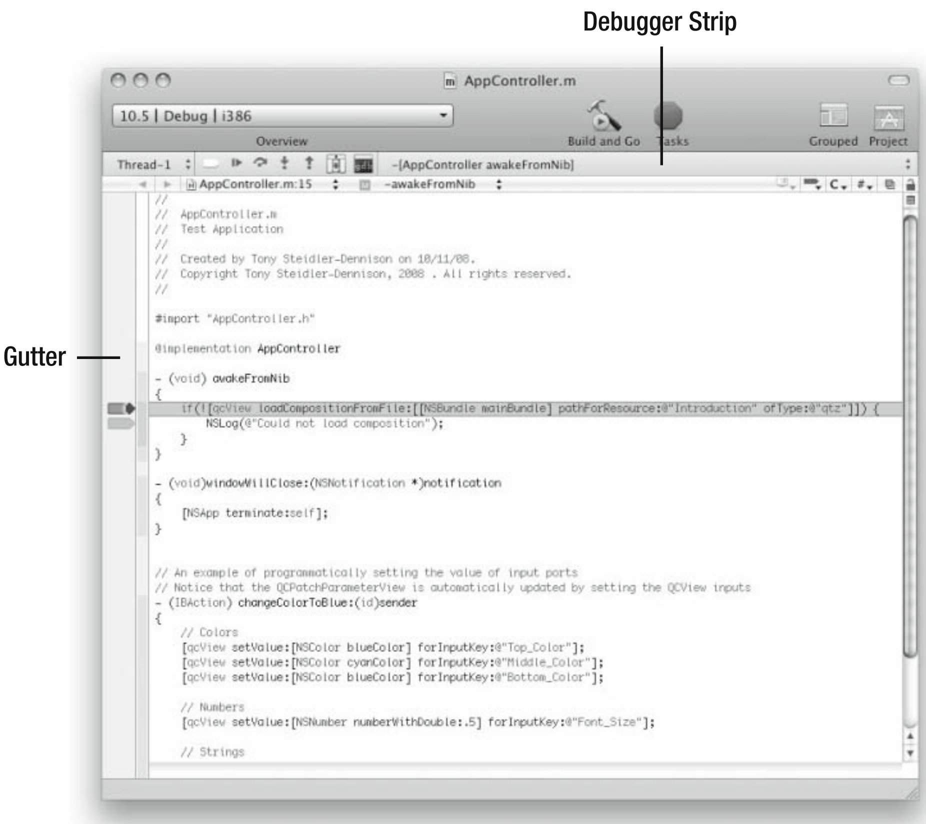
In addition to the project coding and creation tools provided by Xcode, you'll also find a robust debugger. You can step through your own code line by line, or you can attach to and debug a running process that you did not initiate. And, with your own applications, you can set the debugger to attach to any process launched from Xcode only when it crashes.

Consistent with Apple's philosophy of creating tools that are flexible for developers, Xcode provides a number of ways to debug your applications. The method you choose is entirely up to you, and will undoubtedly depend on your preferred work style and environment.

Running the Debugger in the Text Editor

The Xcode debugger provides the ability to debug your code directly in the text editor. This can be a big time-saver if you're creating and debugging code on the fly. To access the debugger in the text editor, double-click the file to be debugged to open it in the text editor. Then, with the text editor in the foreground, select Run ➤ Go (Debug) from the Xcode menu.

Two pieces of the debugger in the text editor are important to you, as noted in Figure 8-13: the debugger strip and the gutter. The debugger strip, shown in Figure 8-14, includes the items listed in Table 8-1. The gutter allows you to set or edit breakpoints.



```
//
//  AppController.m
//  Test Application
//
//  Created by Tony Steidler-Dennison on 10/11/08.
//  Copyright Tony Steidler-Dennison, 2008 . All rights reserved.
//

#import "AppController.h"

@implementation AppController

- (void) awakeFromNib
{
    if(![qcView loadCompositionFromFile:[[NSBundle mainBundle] pathForResource:@"Introduction" ofType:@"qtz"]]) {
        NSLog(@"Could not load composition");
    }
}

- (void)windowWillClose:(NSNotification *)notification
{
    [NSApp terminate:self];
}


// An example of programmatically setting the value of input ports
// Notice that the QCPatchParameterView is automatically updated by setting the QCView inputs
- (IBAction) changeColorToBlue:(id)sender
{
    // Colors
    [qcView setValue:[NSColor blueColor] forInputKey:@"Top_Color"];
    [qcView setValue:[NSColor cyanColor] forInputKey:@"Middle_Color"];
    [qcView setValue:[NSColor blueColor] forInputKey:@"Bottom_Color"];

    // Numbers
    [qcView setValue:[NSNumber numberWithDouble:.5] forInputKey:@"Font_Size"];

    // Strings
```

Figure 8-13. *Debugging code in the Xcode text editor*

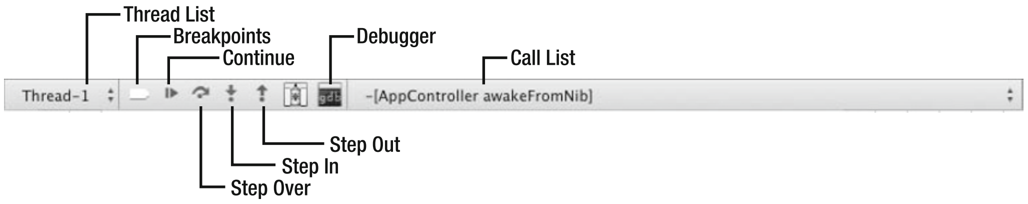


Figure 8-14. *The Xcode debugger strip*

Table 8-1. *Xcode Debugger Strip Items*

Item	Description
Thread list	Shows the thread currently under control of the debugger
Breakpoints button	Sets and deactivates the debugging breakpoints
Continue button	Continues the execution of the debugger
Step Over button	Instructs the debugger to skip, or "step over," the current line of code
Step In button	Instructs the debugger to step into a function or a specific line of code
Step Out button	Instructs the debugger to step out of a function or specific line of code
Debugger button	Opens the GDB (GNU Project Debugger) window
Call list	Shows the call stack (list of called routines)

Additionally, the debugger provides code data tips. By hovering the cursor over the code in debugging mode, you'll have access to a progressive disclosure mechanism that allows you to view and change your application's variables.

Using the Mini Debugger

The full debugging interface, while powerful, can sometimes be a bit of overkill. It's not always necessary to open a full window, nor is it desirable when what you really need is just a quick assessment of how your program is operating. For those purposes, Xcode provides a mini debugger. This small debugging interface floats above a running application, providing many of the same tools that are available in the full version.

The mini debugger, shown in Figure 8-15, includes functions to (from left to right) stop and pause the code, select the project, and activate/deactivate breakpoints within the code.

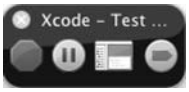
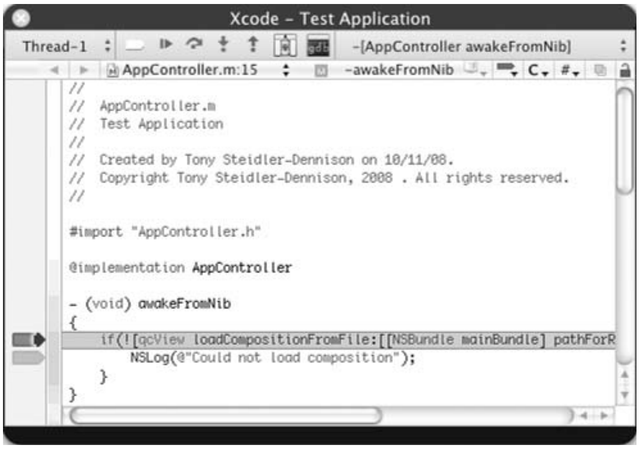


Figure 8-15. *The Xcode mini debugger*

Figure 8-16 shows an example of the mini debugger in operation. In this figure, the process under control of the debugger, known as the *inferior*, is stopped.



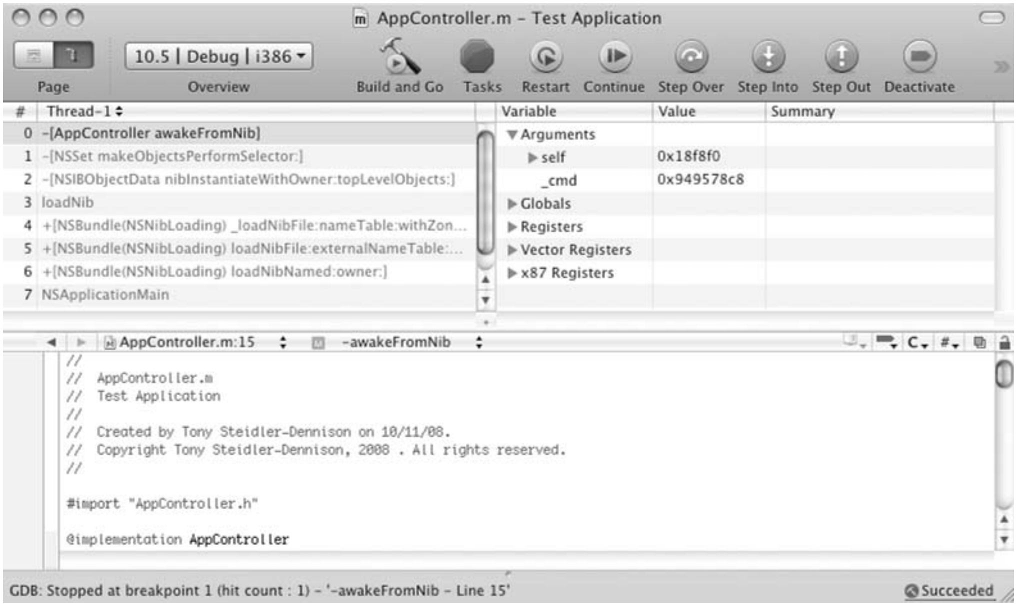
```
//
//  AppController.m
//  Test Application
//
//  Created by Tony Steidler-Dennison on 10/11/08.
//  Copyright Tony Steidler-Dennison, 2008 . All rights reserved.
//

#import "AppController.h"

@implementation AppController

- (void) awakeFromNib
{
    if(![qcView loadCompositionFromFile:[[NSBundle mainBundle] pathForR
        NSLog(@"Could not load composition");
    }
}
```

Figure 8-16. *The mini debugger in operation*

The mini debugger offers many of the same tools as the text editor debugger, including the debugger strip and the gutter. However, unlike the text editor, the mini debugger doesn't allow changes to the source files.

Using the Debugger Window

Xcode provides another interface for debugging to accompany the text editor and the mini debugger. The debugger window, shown in Figure 8-17, is the full debugging interface in Xcode.



#	Thread-1
0	-[AppController awakeFromNib]
1	-[NSSet makeObjectsPerformSelector:]
2	-[NSIBObjectData nibInstantiateWithOwner:topLevelObjects:]
3	loadNib
4	+[NSBundle(NSNibLoading) _loadNibFile:nameTable:withZon…
5	+[NSBundle(NSNibLoading) loadNibFile:externalNameTable:…
6	+[NSBundle(NSNibLoading) loadNibNamed:owner:]
7	NSApplicationMain

Variable	Value	Summary
▼Arguments		
▶self	0x18f8f0	
_cmd	0x949578c8	
▶Globals		
▶Registers		
▶Vector Registers		
▶x87 Registers		

```
//
//  AppController.m
//  Test Application
//
//  Created by Tony Steidler-Dennison on 10/11/08.
//  Copyright Tony Steidler-Dennison, 2008 . All rights reserved.
//

#import "AppController.h"

@implementation AppController
```

```
GDB: Stopped at breakpoint 1 (hit count : 1) – '-awakeFromNib – Line 15'
```

Figure 8-17. *The Xcode debugger window*

The debugger window includes the full set of debugging tools available in Xcode, as listed in Table 8-2 and shown in Figure 8-18.

Table 8-2. *Xcode Debugger Window Toolbar Options*

Button	Description
Build and Go	Builds and runs the application
Tasks (Stop)	Terminates the inferior
Activate/Deactivate	Toggles breakpoints
Fix	Builds a single file fix
Restart	Runs the application in the same state as the previous run
Pause/Continue	Pauses/continues application execution
Step Over	Steps over the current line of code
Step Into	Steps into the call to the current line
Step Out	Steps out of the current function or method

Button	Description
Breakpoints +	Adds a breakpoint
Breakpoints	Opens the breakpoint window
Console	Opens the Console window

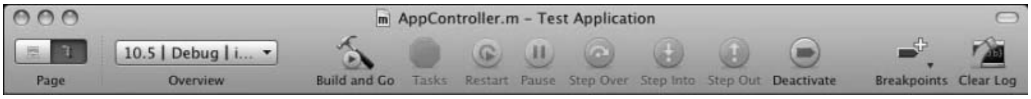


Figure 8-18. *The debugger window toolbar*

With multiple panes, the debugger window provides a wealth of other execution and debugging information. The upper-left pane contains the thread list, with the call stack of the current thread. The upper-right pane contains the variables list, displaying the variables defined in the current scope and any associated values. The lower pane contains the text editor. The status bar resides at the bottom of the window itself, just beneath the text editor. The debugger window also provides some display flexibility, with configuration options for both horizontal (as shown in Figure 8-17) and vertical display.

In short, Xcode provides a powerful set of tools for debugging your application, and maximum flexibility in how those tools are configured and used.

Xcode and Other Application Development Tools

Even though Xcode is the chosen tool for most developers creating applications specifically for Mac OS X, it's not limited to Objective-C, C++, or Java. It is, in fact, a thoroughly modern tool with the flexibility to make full use of other current programming and scripting languages. Whether you're a coder developing object-oriented applications in Python, a web guru creating database-driven sites with Ruby, or a developer who has chosen PHP as your preferred programming language, Xcode will be useful.

The benefits of using Xcode as your development tool might not be obvious until you've used Mac OS X itself for awhile. Xcode provides the native Mac OS X environment, including proper keyboard shortcuts and controls. It also "understands" Subversion, providing both development and source control within a single tool.

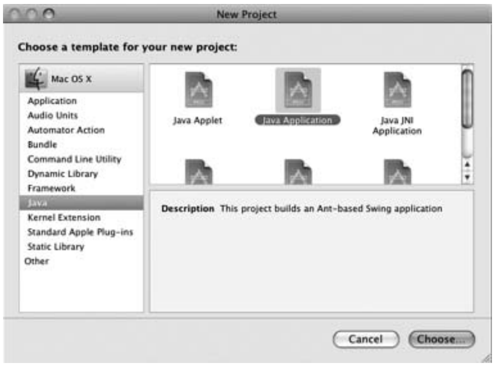
The following sections introduce Java, Python, Ruby, and PHP development with Xcode.

■ **Note** Xcode provides a strong set of tools for object-oriented programming and scripted solutions. *Xcode Unleashed* by Fritz Anderson (Sams, 2008) and *Beginning Xcode* by James Bucanek (Wrox, 2006) provide full, book-length views of Xcode.

Xcode and Java

Java is another development language included in Mac OS X. Java 2 Standard Edition 5 (J2SE 5) is included in the standard Mac OS X installation, with J2SE 6 available as a software update. Both 32- and 64-bit versions of Java are included in the J2SE 5 installation, while the J2SE 6 version is 64-bit and Intel only.

It's easy to create a new Java project in Xcode, as shown in Figure 8-19. As Java is a native development language in Mac OS X, no further modification is required for Xcode to see and make Java available within its tool set.



Description This project builds an Ant-based Swing application

Figure 8-19. *Creating a Java application in Xcode*

Some additional tools are installed in Mac OS X specifically for use with Java. Apache Ant is the tool used by Mac OS X to compile and run Java applications. This, too, is included in the standard Mac OS X installation. The Jar Bundler allows developers to build and deploy Java JAR files as applications that can be launched in the same way as any other Mac OS X application. These JAR files won't require the use of the terminal for operation. Additionally, the Mac OS X installation includes Applet Launcher, which simplifies applet testing in Mac OS X by providing a GUI to Sun's Java plug-in. Applets can be launched from an HTML page, with applet performance and behavior settings configurable via the Java Preferences application.

Mac OS X contains some additional Java-specific development and deployment tools, including the following:

- *Java Web Start*: A tool to launch and modify settings for Java Web Start applications.
- *Java Preferences*: A tool that allows developers to specify settings for Java applications, plug-ins, and applets.
- *Input Method HotKey*: A tool that lets developers set a keyboard combination for invoking the input method dialog box in applications with multiple input methods.
- *JUnit*: A Java unit testing interface.
- *Apache Maven*: A development consolidation tool, including dependency and release management.
- *Apache Ant*: A tool to automate Java builds.

Xcode and Python

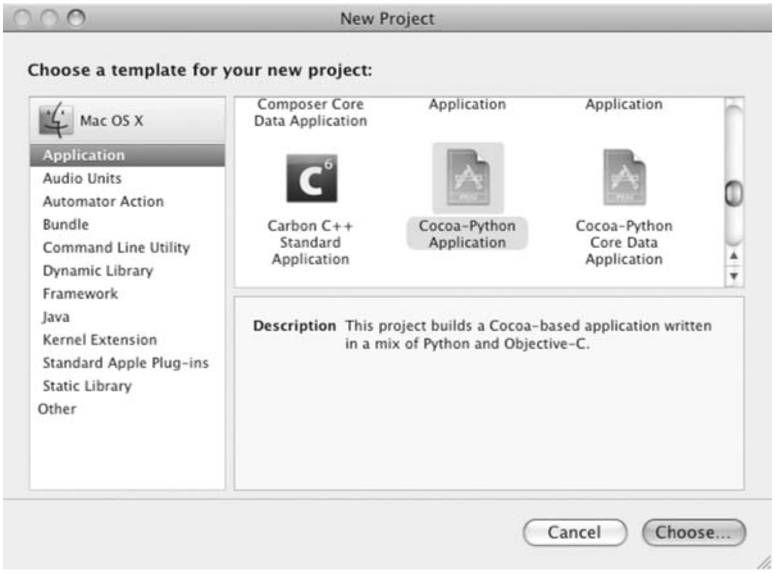
Python, the programming language creation of Guido von Rossum in 1990, has become increasingly popular in the past several years. Renowned for its clean syntax, reasonable learning curve, extensibility, and full object-orientation, Python has garnered a growing following of developers. It's often used as a scripting language to meet quick, one-off needs. Its use has also broadened to include 3D animation and rendering packages such as Maya, graphics creation and manipulation applications such as GIMP and Inkscape, and even games, including Civilization IV.

Released under a GPL-compatible license, Python has also garnered a large and robust user community. It's now a standard element in most Linux distributions. It's also included in Mac OS X, with the Python packages listed in Table 8-3 installed by default.

Table 8-3. *Python Packages Included with Mac OS X Installation*

Package	Description
altgraph	Python graph (network) package
bdist_mpkg	Tool for building Mac OS X installer packages from `distutils`
macholib	Mach-O header analysis and editing
modulegraph	Python module dependency analysis tool
NumPy	Array processing for numbers, strings, records, and objects
py2app	Tool for creating stand-alone Mac OS X applications with Python
setuptools	Utility to download, build, install, upgrade, and uninstall Python packages
xattr	Python wrapper for Darwin's extended filesystem attributes
Twisted	Event-driven networking engine
wxPython	Python bindings for the wxWidgets tool kit
Zope	Open source application server

When the `.mpkg` or source-built installation is complete, Xcode will recognize PyObjC, allowing you to create a new Python project from the menu. As shown in Figure 8-20, Xcode provides the option to create new Python Cocoa projects directly from the New Project window.



Description This project builds a Cocoa-based application written in a mix of Python and Objective-C.

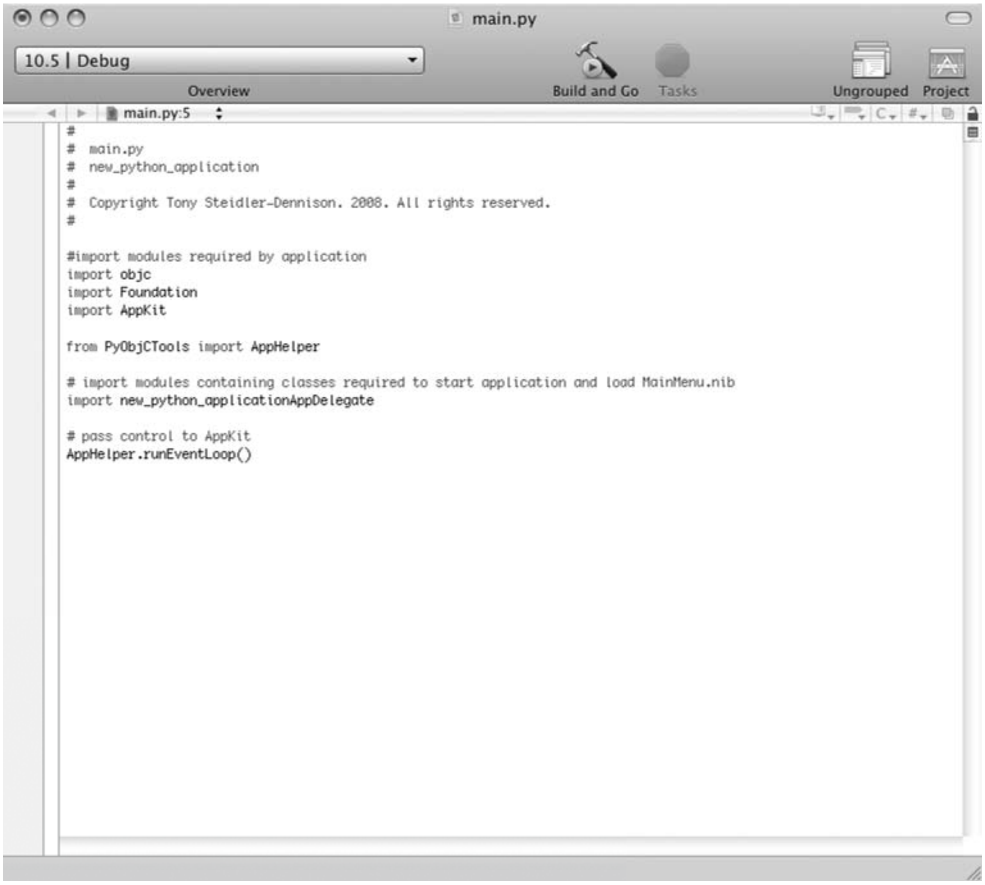
Figure 8-20. *Creating a new Python project in Xcode*

As part of the new Python Cocoa project, the Xcode tool creates the `main.py` file, as shown in Figure 8-21. The tool writes appropriate includes to the file, such as `objc` and Mac OS X classes, including `Foundation` and `AppKit`. Notice that syntax highlighting is fully functional with the PyObjC bindings in Xcode.

While all the Xcode project management tools are available for Python projects, the Xcode debugging tools do not work for Python applications. Python does, however, include the built-in `pdb` debugging module. This is a robust debugging tool, executed from the command line, as follows:

```
$ python -m pdb main.py
> /Users/tony/Development/new_python_application/main.py(10)<module>()
-> import objc
(Pdb)
```

A useful overview of the `pdb` functions is available on the Python site at `http://docs.python.org/library/pdb.html`.



```
#
#  main.py
#  new_python_application
#
#  Copyright Tony Steidler-Dennison. 2008. All rights reserved.
#

#import modules required by application
import objc
import Foundation
import AppKit

from PyObjCTools import AppHelper

# import modules containing classes required to start application and load MainMenu.nib
import new_python_applicationAppDelegate

# pass control to AppKit
AppHelper.runEventLoop()
```

Figure 8-21. *The main.py file in the Xcode text editor*

Xcode and Ruby

Development with Ruby in Xcode is similar to Python development. Like Python, Ruby is included in the base installation of Mac OS X. This also includes Rails, the chosen framework for most current Ruby application development. A number of other Ruby utilities—Ruby gems—are included in the base installation, as well. Overall, Ruby developers will find the Mac OS X Ruby implementation to be extremely friendly and well devised. If you're already familiar with the general user interface layout of the Mac operating system, moving your Ruby development to this platform should be a painless process.

Like Python, Mac OS X provides a robust Ruby installation, including the packages listed in Table 8-4.

Table 8-4. *Ruby Packages Included with Mac OS X Installation*

Package	Description
RubyGems	Ruby package manager
rake	Make-like utility for Ruby scripts
Rails	Framework for database-backed web applications
Mongrel	HTTP library and server, used primarily to build and test Ruby applications
Capistrano	Framework and utility for executing commands in parallel on multiple remote machines, via SSH
Ferret	Search engine
OpenID	Service that provides OpenID identification to Ruby programs
sqlite3-ruby	Module that enables Ruby scripts to interact with a SQLite 3 database
libxml-ruby	Module to read and write XML documents using Ruby
dnssd	Ruby interface for DNS service discovery, implemented as Bonjour in Mac OS X
net	Pure Ruby implementations of the SSH and SFTP client protocols

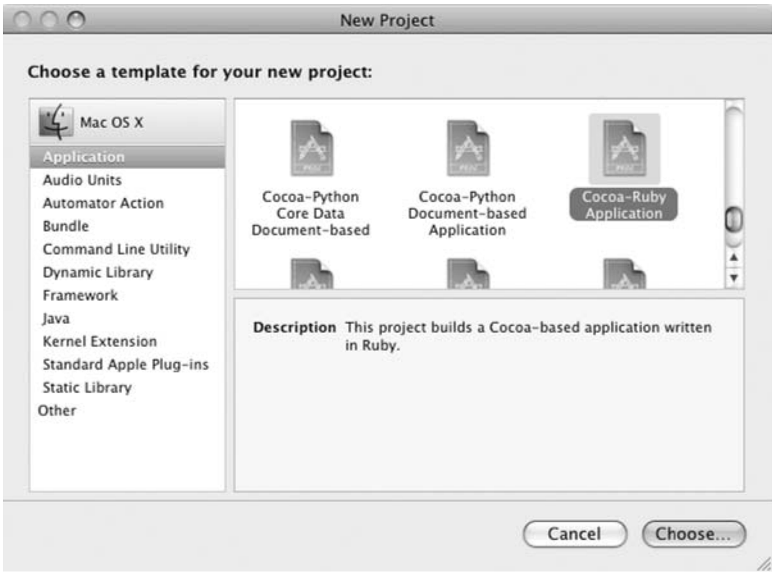
Additional Ruby libraries and modules can be downloaded from `http://rubyforge.org/`. These are available in `.tgz` source code packages or as `.gem` files, available to RubyGems.

Using the RubyGems tool, a developer can add libraries and packages developed by other Ruby users. This follows the model created by Perl developers and the CPAN system. In short, these libraries represent a true implementation of modular design. With a good understanding and frequent use of the modules found on the RubyForge site, you'll clearly save development time, effort, and debugging by utilizing prewritten code.

As with PyObjC, the installation of RubyCocoa will make Ruby visible to the Xcode tool. As shown in Figure 8-22, it's possible to create a new Ruby Cocoa project directly from Xcode with the RubyCocoa framework installed.

■ **Note** The RubyCocoa site (`http://rubycocoa.sourceforge.net/HomePage`) provides ample resources to get you started with Ruby development in Mac OS X., including articles on "the Ruby way," Ruby extensions, and detailed tutorials on Cocoa programming with Ruby. An even more detailed list of RubyCocoa resources can be found at the Ruby Inside site (`http://www.rubyinside.com/the-ultimate-list-of-rubycocoa-tutorials-tips-and-tools-728.html`).

It's clear that the UNIX underpinnings of Mac OS X provide much the same flexibility for Python and Ruby development as that found in most Linux distributions. While a few extra steps may be required to configure a Mac OS X system for Cocoa development with Python or Ruby, the basic functionality of both exists in the standard installation.



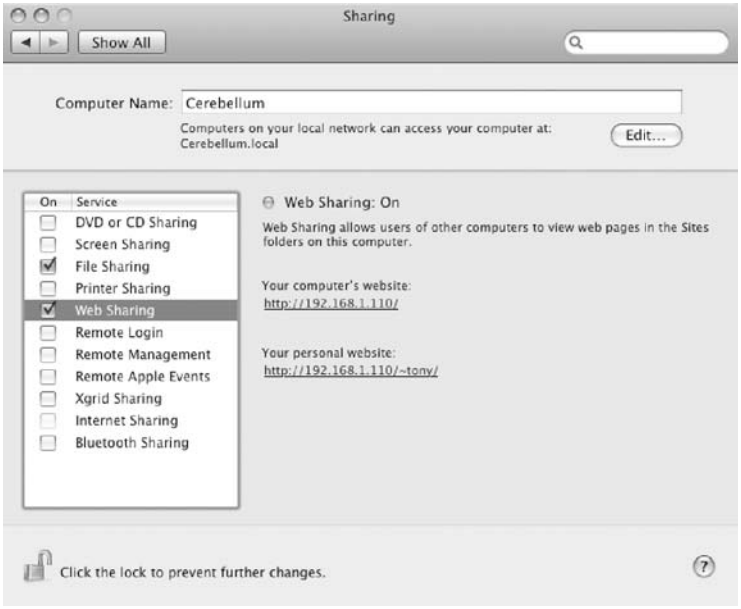
Description This project builds a Cocoa-based application written in Ruby.

Figure 8-22. *Creating a new RubyCocoa application project with Xcode*

Xcode and PHP

Given the inclusion of Java, Python, and Ruby in the standard Mac OS X installation, it should come as no surprise that PHP is also included in Mac OS X. Mac OS X 10.5 (Leopard) installs PHP 5 by default, with built-in support for the SQLite database. The inclusion of PHP and SQLite, in combination with the default Apache installation, makes Mac OS X a strong web application development environment, requiring little additional configuration.

PHP configuration in Mac OS X starts with the setup of the built-in Apache server. To turn on the server, select Sharing from System Preferences, and check the Web Sharing check box, as shown in Figure 8-23. This enables the Apache server on your Mac OS X machine, using both a system home page and a user-specific home page, as noted in the links within the configuration window. You can check the status of the server by clicking the home page links in this window. As shown in Figure 8-24, a default home page is displayed in Mac OS X when the Apache server is properly configured. The index file is located in `/Users/[user]/Sites`.



Computer Name: Cerebellum

Computers on your local network can access your computer at: Cerebellum.local

On	Service
☐	DVD or CD Sharing
☐	Screen Sharing
✓	File Sharing
☐	Printer Sharing
✓	Web Sharing
☐	Remote Login
☐	Remote Management
☐	Remote Apple Events
☐	Xgrid Sharing
☐	Internet Sharing
☐	Bluetooth Sharing

Web Sharing: On

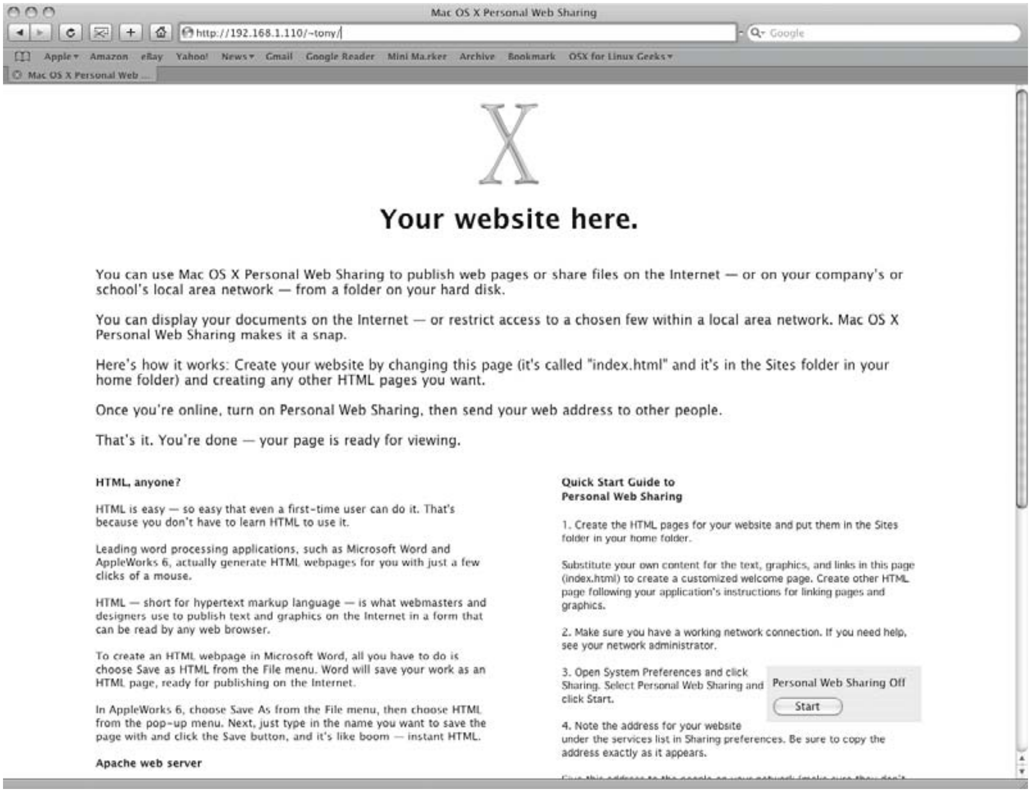
Web Sharing allows users of other computers to view web pages in the Sites folders on this computer.

Your computer's website:
http://192.168.1.110/

Your personal website:
http://192.168.1.110/~tony/

Click the lock to prevent further changes.

Figure 8-23. *Configuring the Apache server from System Preferences*


Mac OS X Personal Web Sharing
http://192.168.1.110/~tony/
Google
Apple▾ Amazon eBay Yahoo! News▾ Gmail Google Reader Mini Ma.rker Archive Bookmark OSX for Linux Geeks▾
Mac OS X Personal Web ...
X

Your website here.

You can use Mac OS X Personal Web Sharing to publish web pages or share files on the Internet — or on your company's or school's local area network — from a folder on your hard disk.

You can display your documents on the Internet — or restrict access to a chosen few within a local area network. Mac OS X Personal Web Sharing makes it a snap.

Here's how it works: Create your website by changing this page (it's called "index.html" and it's in the Sites folder in your home folder) and creating any other HTML pages you want.

Once you're online, turn on Personal Web Sharing, then send your web address to other people.

That's it. You're done — your page is ready for viewing.

HTML, anyone?

HTML is easy — so easy that even a first-time user can do it. That's because you don't have to learn HTML to use it.

Leading word processing applications, such as Microsoft Word and AppleWorks 6, actually generate HTML webpages for you with just a few clicks of a mouse.

HTML — short for hypertext markup language — is what webmasters and designers use to publish text and graphics on the Internet in a form that can be read by any web browser.

To create an HTML webpage in Microsoft Word, all you have to do is choose Save as HTML from the File menu. Word will save your work as an HTML page, ready for publishing on the Internet.

In AppleWorks 6, choose Save As from the File menu, then choose HTML from the pop-up menu. Next, just type in the name you want to save the page with and click the Save button, and it's like boom — instant HTML.

Apache web server

Quick Start Guide to Personal Web Sharing

1. Create the HTML pages for your website and put them in the Sites folder in your home folder.

Substitute your own content for the text, graphics, and links in this page (index.html) to create a customized welcome page. Create other HTML page following your application's instructions for linking pages and graphics.

2. Make sure you have a working network connection. If you need help, see your network administrator.

3. Open System Preferences and click Sharing. Select Personal Web Sharing and click Start.

Personal Web Sharing Off

Start

4. Note the address for your website under the services list in Sharing preferences. Be sure to copy the address exactly as it appears.

Figure 8-24. *The default user home page in Mac OS X*

While the Apache server is configured out of the box, configuring PHP requires a few additional steps. To load the PHP module in Apache at startup, uncomment the following line in `/etc/apache2/httpd.conf`:

```
#LoadModule php5_module        libexec/apache2/libphp5.so
```

Then restart the Apache server with the following command:

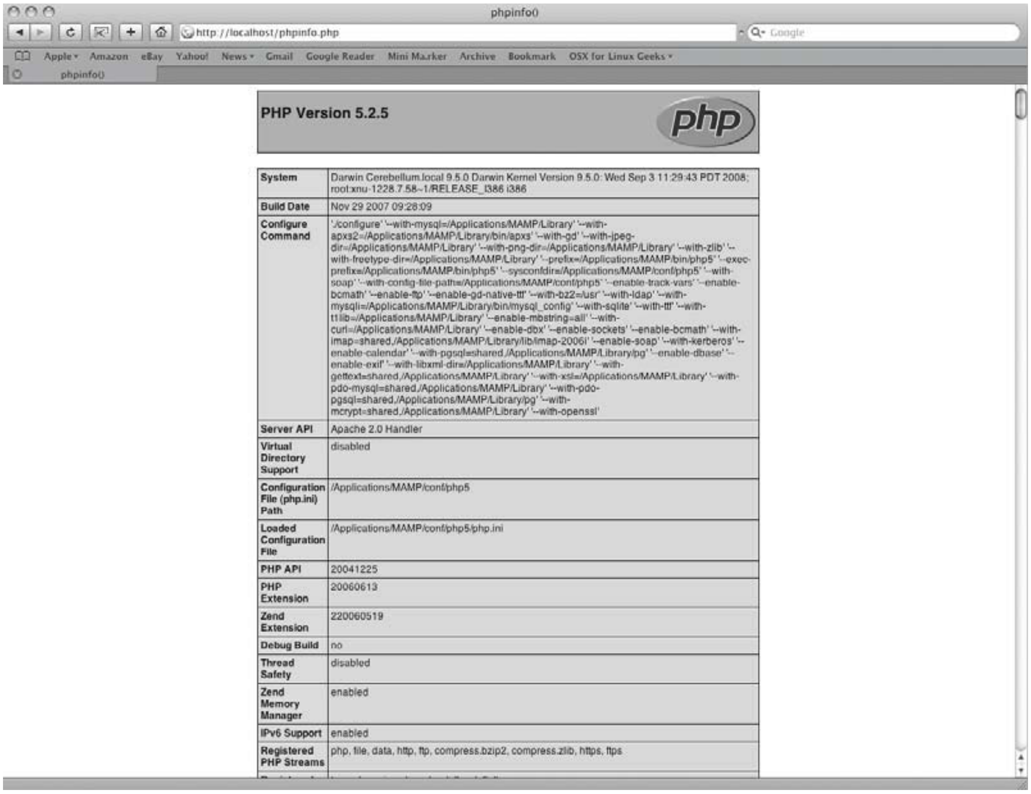
```
$ sudo /usr/sbin/apachectl restart
```

This will restart the server, loading the PHP 5 module.

You can check the PHP installation by creating a file in the server directory. Simply create a file named `phpinfo.php` in the document root containing the following:

```
<?php
phpinfo();
?>
```

By loading this page in your browser, as shown in Figure 8-25, you'll test the PHP configuration, as well as display all the pertinent configuration information.


phpinfo()
http://localhost/phpinfo.php
Google
Apple Amazon eBay Yahoo! News Gmail Google Reader Mini Marker Archive Bookmark OSX for Linux Geeks
phpinfo()
php

PHP Version 5.2.5

System	Darwin Cerebellum.local 9.5.0 Darwin Kernel Version 9.5.0: Wed Sep 3 11:29:43 PDT 2008; root:xnu-1228.7.58~1/RELEASE_I386 i386
Build Date	Nov 29 2007 09:28:09
Configure Command	'./configure' '--with-mysql=/Applications/MAMP/Library' '--with-apxs2=/Applications/MAMP/Library/bin/apxs' '--with-gd' '--with-jpeg-dir=/Applications/MAMP/Library' '--with-png-dir=/Applications/MAMP/Library' '--with-zlib' '--with-freetype-dir=/Applications/MAMP/Library' '--prefix=/Applications/MAMP/bin/php5' '--exec-prefix=/Applications/MAMP/bin/php5' '--sysconfdir=/Applications/MAMP/conf/php5' '--with-soap' '--with-config-file-path=/Applications/MAMP/conf/php5' '--enable-track-vars' '--enable-bcmath' '--enable-ftp' '--enable-gd-native-ttf' '--with-bz2=/usr' '--with-ldap' '--with-mysqli=/Applications/MAMP/Library/bin/mysql_config' '--with-sqlite' '--with-ttf' '--with-t1lib=/Applications/MAMP/Library' '--enable-mbstring=all' '--with-curl=/Applications/MAMP/Library' '--enable-dbx' '--enable-sockets' '--enable-bcmath' '--with-imap=shared,/Applications/MAMP/Library/lib/imap-2006i' '--enable-soap' '--with-kerberos' '--enable-calendar' '--with-pgsql=shared,/Applications/MAMP/Library/pg' '--enable-dbase' '--enable-exif' '--with-libxml-dir=/Applications/MAMP/Library' '--with-gettext=shared,/Applications/MAMP/Library' '--with-xsl=/Applications/MAMP/Library' '--with-pdo-mysql=shared,/Applications/MAMP/Library' '--with-pdo-pgsql=shared,/Applications/MAMP/Library/pg' '--with-mcrypt=shared,/Applications/MAMP/Library' '--with-openssl'
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/Applications/MAMP/conf/php5
Loaded Configuration File	/Applications/MAMP/conf/php5/php.ini
PHP API	20041225
PHP Extension	20060613
Zend Extension	220060519
Debug Build	no
Thread Safety	disabled
Zend Memory Manager	enabled
IPv6 Support	enabled
Registered PHP Streams	php, file, data, http, ftp, compress.bzip2, compress.zlib, https, ftps

Figure 8-25. *The phpinfo.php page displayed in Safari*

■**Note** PHP development can be greatly enhanced by the use of PEAR modules, available at `http://pear.php.net/`. Like CPAN and RubyGems, PEAR modules are prebuilt code chunks, written to accomplish a specific task. These can be easily rolled into your PHP development. Full documentation to acquire and use PEAR modules is available on the PEAR site at `http://pear.php.net/manual/`.

The inclusion of PHP in the base Mac OS X installation rounds out the remarkably complete set of development tools built into the Mac operating system. You've seen that the Mac OS X platform includes the most current programming and scripting languages, all of which are available (some with minor additional configuration) from within the Xcode interface.

Scripting

Aside from the built-in programming languages, Mac OS X supplies a great environment for scripting and scripting solutions. The UNIX basis of the operating system provides all the tools necessary for shell scripting. As with a Linux system, you can create scripts on the fly to accomplish any number of administrative tasks. Scripts can also provide some functionality in other application development.

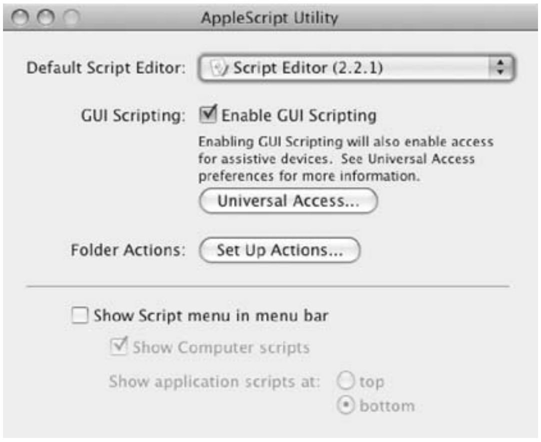
Bash, Python, Perl, and Ruby provide strong scripting functionality, and all are available in the Mac OS X installation. In practice, scripting in Mac OS X using these languages will be virtually indistinguishable from scripting on a Linux or UNIX system. But for many, scripting in Mac OS X starts with the native scripting tool: AppleScript.

Using AppleScript

AppleScript is a scripting language that can respond to a number of events in Mac OS X by performing a set of defined operations or by providing data. An *event* in Mac OS X is an internal message containing commands or arbitrary data. The Open Scripting Architecture (OSA) is the API at the heart of AppleScript. OSA makes it possible to communicate with other scripting languages and with other applications on the system.

Lexically, AppleScript is simple, utilizing 103 reserved keywords. As with Python, the syntax is also simple, making AppleScript an easy language to learn. However, the simplicity of the language itself is deceptive. AppleScript is a rich, object-oriented scripting language, perfectly suited to creating scriptable applications, performing repetitive operations, and providing access to applications or other scripting languages in the system. Apple provides a comprehensive guide to using AppleScript at `http://developer.apple.com/referencelibrary/GettingStarted/GS_AppleScript/index.html`.

The AppleScript Utility, located in the `Applications/Applescript` directory, is used to configure the use of the AppleScript tools on your system. As shown in Figure 8-26, the configuration options include a choice between scripting editor versions, the ability to utilize GUI scripting and provide universal access (including voice control), and the ability to set up folder actions for AppleScripts. (*Folder actions* are repetitive actions taken on the contents of a folder, such as periodically checking whether the folder contents have changed, and moving any new contents to another folder.)



Default Script Editor: Script Editor (2.2.1)

GUI Scripting: [✓ Enable GUI Scripting]

Enabling GUI Scripting will also enable access for assistive devices. See Universal Access preferences for more information.

Folder Actions:

Show Script menu in menu bar

[✓ Show Computer scripts]

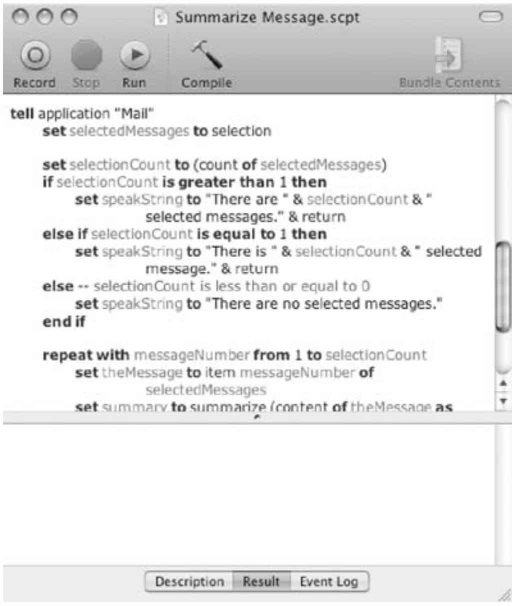
Show application scripts at: top [✓ bottom]

Figure 8-26. *The AppleScript Utility*

Creating Scripts with the Script Editor

Mac OS X provides the Script Editor, located in the `Applications/Applescript` directory. This is a rich editor used primarily for creating, testing and, where necessary, compiling AppleScript scripts. However, it also understands the other scripting languages on the system, including bash, Python, Perl, and Ruby.

As shown in Figure 8-27, the Script Editor utilizes a multipane window and syntax highlighting. The sample script shown in Figure 8-27 is a pure AppleScript script, with the appropriate syntax highlighting. Scripts written in other scripting languages in the Script Editor will be highlighted accordingly.



```
tell application "Mail"
    set selectedMessages to selection

    set selectionCount to (count of selectedMessages)
    if selectionCount is greater than 1 then
        set speakString to "There are " & selectionCount & "
                selected messages." & return
    else if selectionCount is equal to 1 then
        set speakString to "There is " & selectionCount & " selected
                message." & return
    else -- selectionCount is less than or equal to 0
        set speakString to "There are no selected messages."
    end if

    repeat with messageNumber from 1 to selectionCount
        set theMessage to item messageNumber of
                selectedMessages
        set summary to summarize (content of theMessage as
```

Figure 8-27. *The Script Editor*

Figure 8-28 shows the Script Editor toolbar. The buttons on this toolbar allow you to record macros, run and stop a script, and to compile that script into a stand-alone application for Mac OS X.

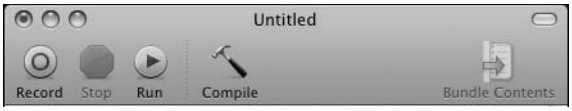


Figure 8-28. *The Script Editor toolbar*

The Script Editor provides a wide range of additional features, including the ability to view the data dictionaries of scriptable applications.

■ **Tip** Also worth noting is Automator, an Apple script-creation tool included in Mac OS X. Automator makes it possible to create scripts without actually writing any code. It's a graphical tool that provides a library of common actions in the form of graphical objects. The interface allows you to create relationships between these actions by dragging and dropping them into the proper sequence. The output is in the form of *workflows*, which carry out the actions specified by the user. Ben Waldie's book *Automator for Mac OS X 10.5 Leopard: Visual QuickStart Guide* (Peachpit Press, 2008) provides a detailed discussion of Automator.

Using Other Scripting Languages

As noted, it's possible to use other programming languages in Mac OS X, just as you would on any other UNIX system. Bash, Perl, Python, and Ruby are all installed with Mac OS X, and are available without further configuration. Scripts in these languages can be created with any text editor or with the Script Editor. The Script Editor also provides GUI-based testing for these scripts. In practice, scripting in Mac OS X using these languages will be virtually indistinguishable from scripting on a Linux or UNIX system.

Code Maintenance and Revision Control

Mac OS X is packed with modern useful development tools: programming languages, a robust debugger, and a project organization and management tool. Combined, they provide a development environment that's very much the equivalent of most Linux systems. Everything necessary to create applications and scripts for all computing platforms is available at no additional cost and only minimal additional effort.

But even a good development environment is incomplete if it doesn't provide a tool for source and revision control. Any developer who has lost a significant chunk of irreplaceable code will vouch for the value of source control. Even lone developers have come to rely increasingly on revision control. The abilities to assess and summarize differences between files, to roll back to previous versions, and to take complete control of all source code are critical to successful software development.

From what you've seen of the built-in tools in Mac OS X, it should come as no surprise that it includes the latest and greatest source control system: Subversion. As you would find in most Linux distributions, Subversion is available both from the command line and with several well-developed front-end tools. Additionally, you can use Git with Mac OS X. Let's look at each of these revision control options, beginning with an overview of Subversion.

Introducing Subversion

Subversion has become a popular tool for version control. Developed by CollabNet in 2000 as a replacement for the Concurrent Versions System (CVS), Subversion has become the primary version control system on open source projects, including Apache, Python, Mono, GNOME, Ruby, and others. It's also moving toward prominence in the commercial world, where tools saddled with annual and per-seat licensing fees have long held sway. Both open source developers and corporate software developers have begun to appreciate the power of Subversion.

Subversion vs. CVS

Subversion offers several important improvements over the older CVS. These improvements provide users with more control over the tool, more flexibility in how versioning is accomplished, and a full set of APIs that make it extremely customizable for unique uses. These improvements are accomplished by the following:

Directory versioning: Subversion utilizes a "virtual" versioned filesystem that allows tracking of both files and directory trees. This is a significant improvement over CVS, which tracked versions only on individual files.

Versioned metadata: The metadata of all files and directories—the data describing the properties of the files and directories—can be created and modified by the user. This data is versioned with the data contained in the files themselves.

Atomic commits: Atomic commits guarantee that if an entire collection of changes cannot be committed to the repository, none are committed. This is important to the work flow of developers, as it allows them to structure their changes in a more logical fashion. It also guarantees that no problems will arise with committed code for which only a partial set of changes have been committed.

Network flexibility: Subversion can be plugged into Apache as a module. It can be used over the network as a stand-alone tool. Subversion can also be implemented within a secure Shell (SSH) tunnel across a wide network.

Binary/text parity: Both binary and text files can be committed and tracked using Subversion. The algorithm implemented to recognize and express the differences between versions is identical in both text and binary files. This results in a much more seamless work flow, in which all files are handled in the same way.

Branching and tagging: Subversion creates branches in a manner similar to hard-linking. (See the "How Does Time Machine Do That?" section of Chapter 7 for a description of hard-linking.) Both branches and tags are created and maintained using this mechanism.

True versioning: One significant drawback of CVS was its inability to distinguish files with the same name. If, for example, a file in a CVS repository was replaced with a new file of the same name, the versioning history of the predecessor file attached to the new file. Though that old versioning history may have literally no relevance to the new file, it was attached. Subversion creates a new version history with each file added to the repository, regardless of a similarity in names. Furthermore, file copies and renames are fully supported in Subversion, unlike CVS.

Open API: Subversion is implemented as a collection of shared C libraries. The APIs for these libraries are well known and well defined. As a result, Subversion can be customized, modified, and extended to more closely suit the users' needs.

Clearly, Subversion is a strong evolution from its predecessor CVS. It provides users with much more power and flexibility than its predecessors, and does so in a much more intelligent way. And, of course, it's included and ready to use in Mac OS X.

Subversion's Copy-Modify-Merge Model

Subversion implements a *copy-modify-merge* model of version control. Most older version control systems, including CVS, utilize a *lock-modify-unlock* model for version control. These models are critical for capturing all changes to a file, even when those files are under concurrent modification by different developers. The problem lies in how those changes are tracked. If the files are simply shared, without either type of version control model in place, changes made by one developer will surely be overwritten by another. That is, in the end, one of the most important reasons to use a version control system, especially in an environment where many developers will have access to a set of files.

In the lock-modify-unlock version control model utilized by CVS and other older version control systems, a file can be modified by only one user at a time. The first user to access the file in the repository "locks" the file, preventing write access by other users. Clearly, this is an inefficient model. One developer must wait for another to finish making changes to a file. Even if the second developer intends to make changes that will not conflict with changes made by the first, she will need to wait until the first user unlocks the file.

The copy-modify-merge model allows individual Subversion users to copy a file from the repository and make changes to the file locally. When complete, those changes will be merged with all other changes made to the file after the time it was checked out. At the time of the commit, Subversion notifies the "last-in" user that additional changes have been made to the file. The developer will then use the `merge` command to modify the local working copy of the file with changes to the file on the repository. If no conflict exists, the "last-in" file is committed seamlessly to the repository. If a conflict does exist between the changes, the user is notified of the conflicts and presented a view of both sets. One set will be selected manually and, once those changes are incorporated, the file can be committed back to the repository.

In short, the copy-modify-merge model is a much more efficient model for tracking modifications to a file. It allows multiple users to truly work on files simultaneously and handles conflicting changes to files in an intelligent fashion.

Using Subversion from the Command Line

Subversion is easy to use from the command line. The syntax is as follows:

```
svn <subcommand> [options] [args]
```

Table 8-5 lists some of the commonly used subcommands. The Subversion help (`svn --help`) lists the full set of subcommands.

Table 8-5. *Common Subversion (svn) Subcommands*

Subcommand	Shortcut	Description
`add`		Adds a new file or directory to an existing Subversion repository
`checkout`	`co`	Gets a local copy of a file or directory from a Subversion repository
`commit`	`ci`	Adds a changed local file back into an existing repository
`diff`	`di`	Provides a list of differences between two file versions
`merge`		Incorporates changes made after the file was checked out into the current version, or displays conflicts
`update`	`up`	Checks out the most current version of a file or directory

Options for the Subversion commands are a bit more esoteric. Subversion options are global, in that each option has the same effect, regardless of the subcommand used. Some of these option/argument pairs include those listed in Table 8-6.

Table 8-6. *Common Subversion (svn) Options*

Option	Description
`--diff-cmd CMD`	Uses an external tool (`CMD`) for diffs
`--editor-cmd CMD`	Uses an external tool (`CMD`) for editing
`--file (-F) FILENAME`	Uses the contents of `FILENAME` to execute subcommands
`--force`	Forces the subcommand to run
`--help`	Displays the `svn` help information
`-- password PASS`	Provides an authentication password on the command line
`--quiet`	Prints only essential information when completing an operation
`--revision (-r) REV`	Manually provides a revision number for the operation; can include a number, keywords, or dates
`--verbose (-v)`	Instructs the client to print as much information as possible while running the subcommand

Subversion also includes the `svnadmin` administrative tool. Like the `svn` command, `svnadmin` utilizes several subcommands, including those listed in Table 8-7.

Table 8-7. *Common Subversion Administration (svnadmin) Subcommands*

Subcommand	Description
`create`	Creates a new Subversion repository
`dump`	Dumps all changes from within a repository (most often used to move a repository from one location to another)
`hotcopy`	Makes a safe copy of a repository, regardless of whether other processes are using the repository
`load`	Loads a set of revisions into a repository, generally from a file created with the `dump` command
`verify`	Verifies the contents of a repository, including checksum comparisons of the data stored in the repository

Using Subversion GUI Front Ends

While command-line Subversion is easily the fastest possible way to utilize it, learning the subcommands and the options may not be your cup of tea. As with many great Linux command-line applications, you'll find a full range of GUI front ends for Subversion on the Mac. These tools are free or very reasonably priced, are easy to learn, and, for the most part, are very much an asset to your use of Subversion. If you're disinclined to use the command line, these tools will still maximize your efficiency and your time in using Subversion for your projects. Here, we'll look at two Subversion GUI front ends: Versions for the Mac and RapidSVN.

Versions for the Mac

Aside from providing a front end for Subversion, Versions for the Mac provides some other interesting features that you won't find in other Subversion clients.

Versions is available at `http://versionsapp.com/`. It is provided as a zip file containing a stand-alone application. It doesn't require any installation other than unzipping the file and dragging the binary into the `Applications` directory. Double-clicking will open Versions.

Figure 8-29 shows the main Versions window. On the first use, Versions provides several options for setting up a new repository or connecting to an existing one.

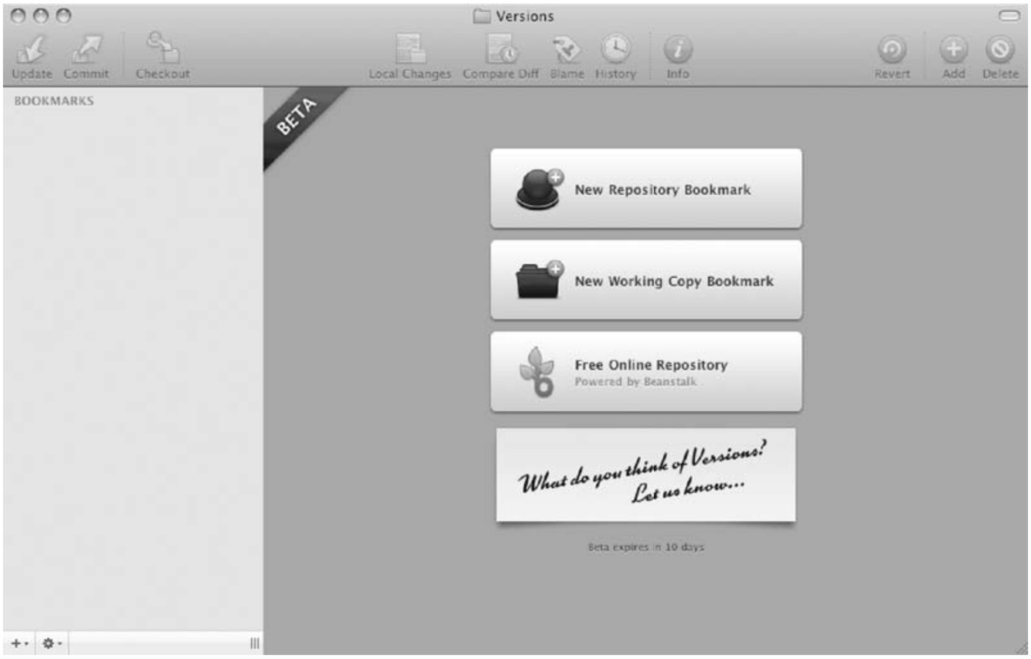
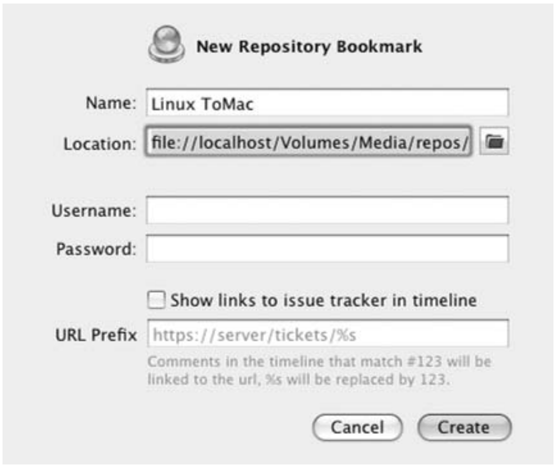


Figure 8-29. *The main Versions window*

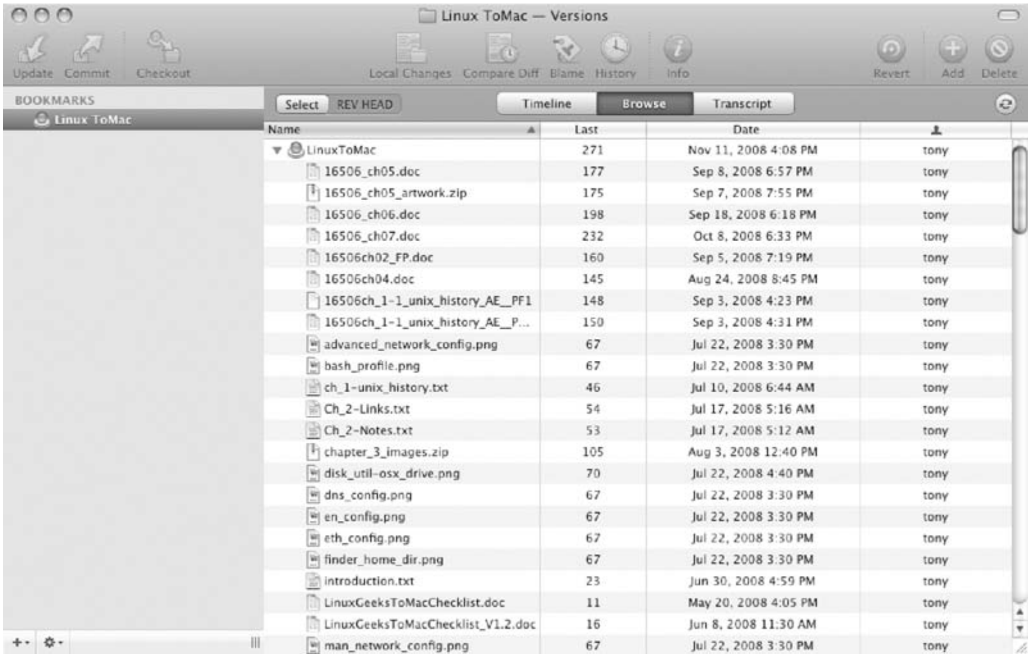
The connections to these repositories are created in the form of bookmarks. Figure 8-30 shows an example of creating a bookmark to an existing Subversion repository (the writing repository named LinuxToMac on my system).


New Repository Bookmark

Name: Linux ToMac
Location: file://localhost/Volumes/Media/repos/
Username:
Password:
☐ Show links to issue tracker in timeline
URL Prefix https://server/tickets/%s
Comments in the timeline that match #123 will be linked to the url, %s will be replaced by 123.

Figure 8-30. *Creating a bookmark to an existing repository in Versions*

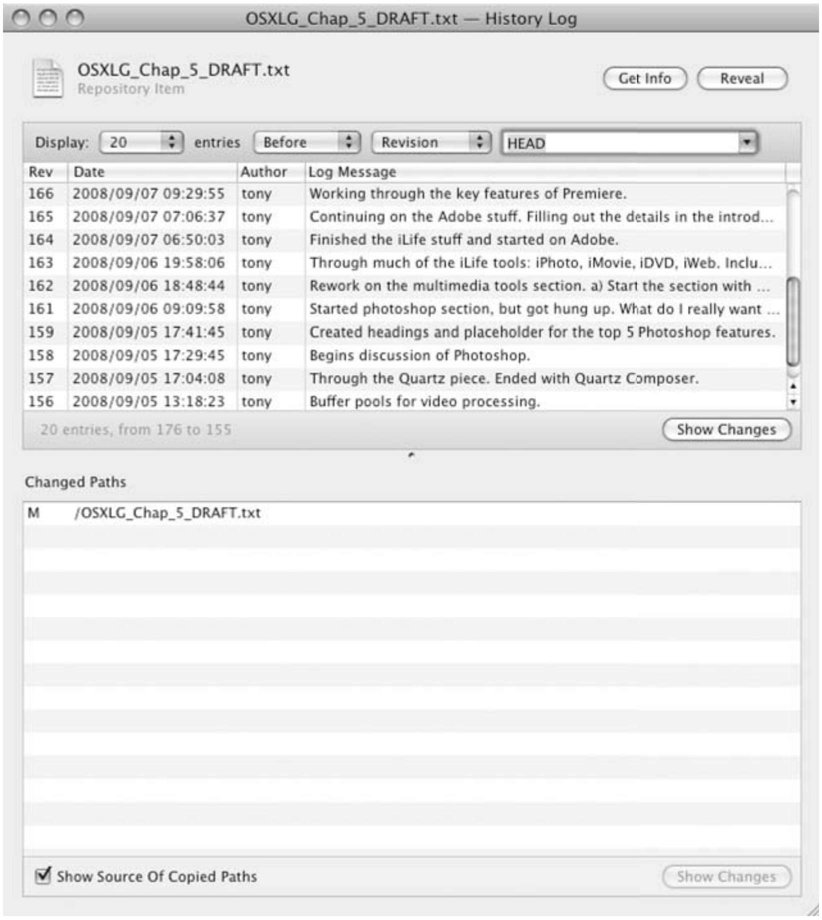
Figure 8-31 displays the contents of the repository accessed by the bookmark created in Figure 8-30. This is the view in the Browse tab of Versions, which shows all the files and directories in the repository.



Name	Last	Date	
LinuxToMac	271	Nov 11, 2008 4:08 PM	tony
16506_ch05.doc	177	Sep 8, 2008 6:57 PM	tony
16506_ch05_artwork.zip	175	Sep 7, 2008 7:55 PM	tony
16506_ch06.doc	198	Sep 18, 2008 6:18 PM	tony
16506_ch07.doc	232	Oct 8, 2008 6:33 PM	tony
16506ch02_FP.doc	160	Sep 5, 2008 7:19 PM	tony
16506ch04.doc	145	Aug 24, 2008 8:45 PM	tony
16506ch_1-1_unix_history_AE__PF1	148	Sep 3, 2008 4:23 PM	tony
16506ch_1-1_unix_history_AE__P...	150	Sep 3, 2008 4:31 PM	tony
advanced_network_config.png	67	Jul 22, 2008 3:30 PM	tony
bash_profile.png	67	Jul 22, 2008 3:30 PM	tony
ch_1-unix_history.txt	46	Jul 10, 2008 6:44 AM	tony
Ch_2-Links.txt	54	Jul 17, 2008 5:16 AM	tony
Ch_2-Notes.txt	53	Jul 17, 2008 5:12 AM	tony
chapter_3_images.zip	105	Aug 3, 2008 12:40 PM	tony
disk_util-osx_drive.png	70	Jul 22, 2008 4:40 PM	tony
dns_config.png	67	Jul 22, 2008 3:30 PM	tony
en_config.png	67	Jul 22, 2008 3:30 PM	tony
eth_config.png	67	Jul 22, 2008 3:30 PM	tony
finder_home_dir.png	67	Jul 22, 2008 3:30 PM	tony
introduction.txt	23	Jun 30, 2008 4:59 PM	tony
LinuxGeeksToMacChecklist.doc	11	May 20, 2008 4:05 PM	tony
LinuxGeeksToMacChecklist_V1.2.doc	16	Jun 8, 2008 11:30 AM	tony
man_network_config.png	67	Jul 22, 2008 3:30 PM	tony

Figure 8-31. *The contents of the repository, accessed by clicking the bookmark in Versions*

By highlighting a file in the Browse tab, you can select from any of the tools in the toolbar in the main window. Figure 8-32 shows the result of selecting the History tool when highlighting the `OSXLG_Chap_5_DRAFT.txt` file. As you can see, the default view is the HEAD view, which reads the commit metadata, including revision, date, author, and messages. Using drop-down menus in the window, you can further refine your view by broadening or narrowing the number of viewable entries. You can also view the revisions by date or other Subversion command.


OSXLG_Chap_5_DRAFT.txt — History Log
OSXLG_Chap_5_DRAFT.txt
Repository Item
Get Info
Reveal
Display: 20 entries Before Revision HEAD
Show Changes
Changed Paths
Show Source Of Copied Paths
Show Changes

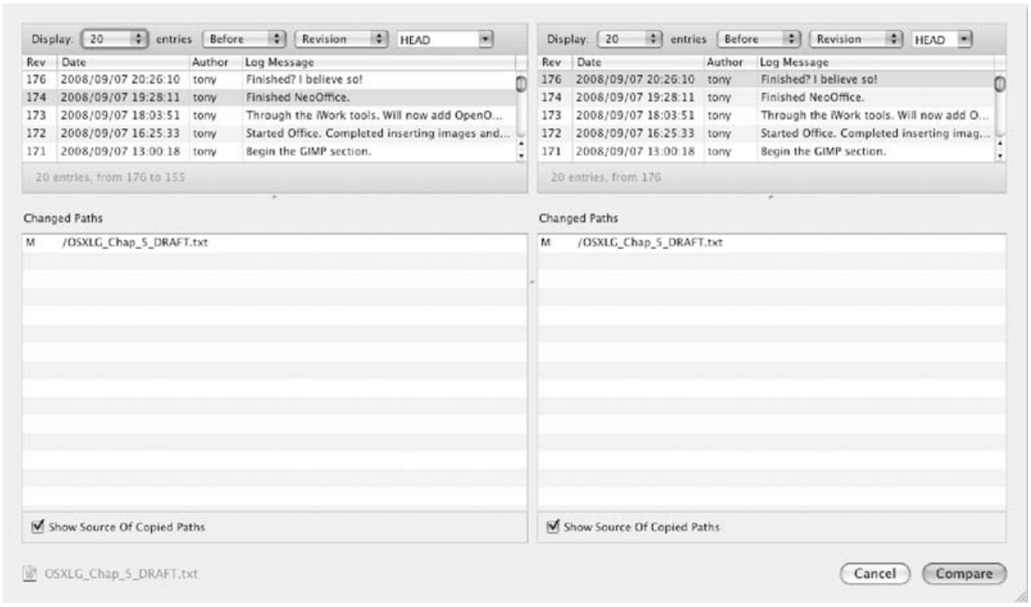
Rev	Date	Author	Log Message
166	2008/09/07 09:29:55	tony	Working through the key features of Premiere.
165	2008/09/07 07:06:37	tony	Continuing on the Adobe stuff. Filling out the details in the introd...
164	2008/09/07 06:50:03	tony	Finished the iLife stuff and started on Adobe.
163	2008/09/06 19:58:06	tony	Through much of the iLife tools: iPhoto, iMovie, iDVD, iWeb. Inclu...
162	2008/09/06 18:48:44	tony	Rework on the multimedia tools section. a) Start the section with ...
161	2008/09/06 09:09:58	tony	Started photoshop section, but got hung up. What do I really want ...
159	2008/09/05 17:41:45	tony	Created headings and placeholder for the top 5 Photoshop features.
158	2008/09/05 17:29:45	tony	Begins discussion of Photoshop.
157	2008/09/05 17:04:08	tony	Through the Quartz piece. Ended with Quartz Composer.
156	2008/09/05 13:18:23	tony	Buffer pools for video processing.

20 entries, from 176 to 155

```
M      /OSXLG_Chap_5_DRAFT.txt
```

Figure 8-32. *Checking the history of a file from the Browse tab in Versions*

Figure 8-33 shows the use of the Compare Diff tool to compare two different versions of a single file. As you can see, the Versions interface puts all commits of a file side by side in the window.



Display: 20 entries Before Revision HEAD

Rev	Date	Author	Log Message
176	2008/09/07 20:26:10	tony	Finished? I believe so!
174	2008/09/07 19:28:11	tony	Finished NeoOffice.
173	2008/09/07 18:03:51	tony	Through the iWork tools. Will now add OpenO...
172	2008/09/07 16:25:33	tony	Started Office. Completed inserting images and...
171	2008/09/07 13:00:18	tony	Begin the GIMP section.

20 entries, from 176 to 155

Changed Paths

M /OSXLG_Chap_5_DRAFT.txt

☑ Show Source Of Copied Paths

Display: 20 entries Before Revision HEAD

Rev	Date	Author	Log Message
176	2008/09/07 20:26:10	tony	Finished? I believe so!
174	2008/09/07 19:28:11	tony	Finished NeoOffice.
173	2008/09/07 18:03:51	tony	Through the iWork tools. Will now add O...
172	2008/09/07 16:25:33	tony	Started Office. Completed inserting imag...
171	2008/09/07 13:00:18	tony	Begin the GIMP section.

20 entries, from 176

Changed Paths

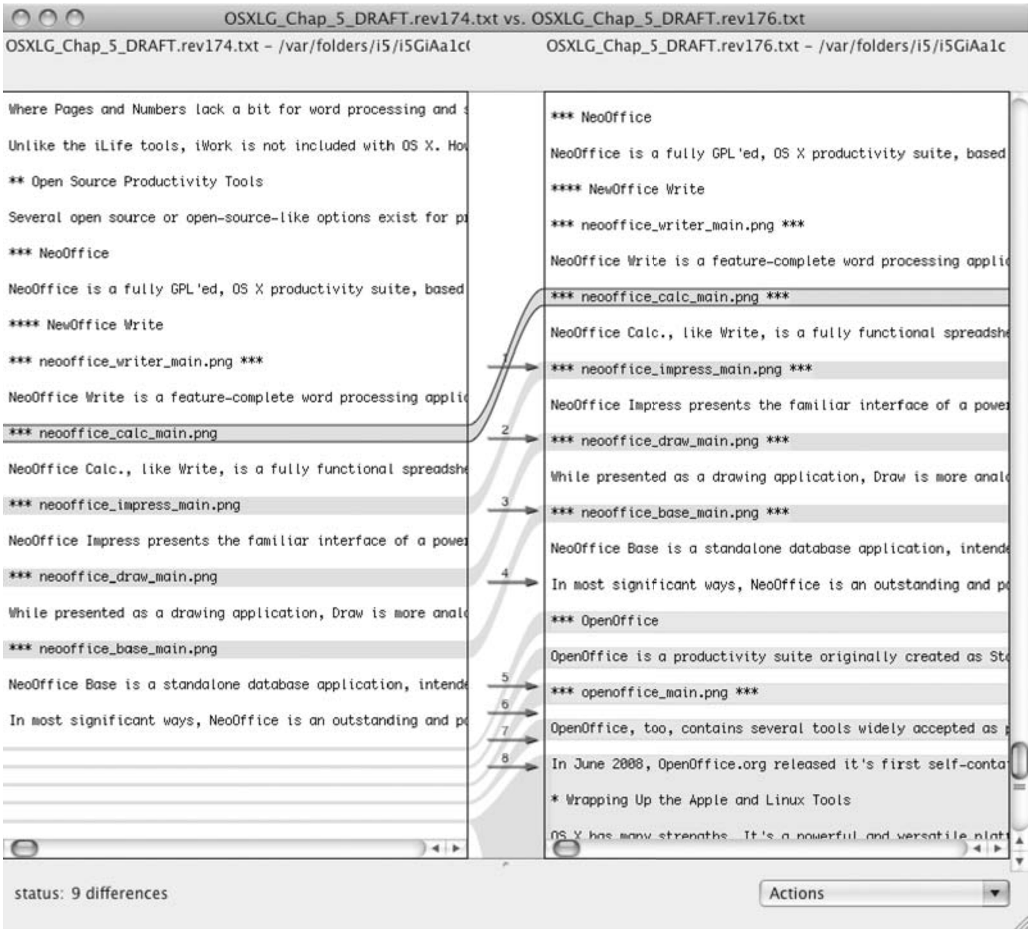
M /OSXLG_Chap_5_DRAFT.txt

☑ Show Source Of Copied Paths

OSXLG_Chap_5_DRAFT.txt

Figure 8-33. *The Compare Diff function in Versions*

Figure 8-34 shows the clean diff presentation provided by Versions. Inserts and deletions are clearly delineated in the Compare Diff window, as is a count of the number of differences. In short, Versions makes it very easy to quickly scan through changes in multiple document versions.


OSXLG_Chap_5_DRAFT.rev174.txt vs. OSXLG_Chap_5_DRAFT.rev176.txt

OSXLG_Chap_5_DRAFT.rev174.txt – /var/folders/i5/i5GiAa1c(

```
Where Pages and Numbers lack a bit for word processing and s
Unlike the iLife tools, iWork is not included with OS X. How
** Open Source Productivity Tools
Several open source or open-source-like options exist for pr
*** NeoOffice
NeoOffice is a fully GPL'ed, OS X productivity suite, based
**** NewOffice Write
*** neooffice_writer_main.png ***
NeoOffice Write is a feature-complete word processing applic
*** neooffice_calc_main.png
NeoOffice Calc., like Write, is a fully functional spreadshe
*** neooffice_impress_main.png
NeoOffice Impress presents the familiar interface of a power
*** neooffice_draw_main.png
While presented as a drawing application, Draw is more analo
*** neooffice_base_main.png
NeoOffice Base is a standalone database application, intende
In most significant ways, NeoOffice is an outstanding and po
```

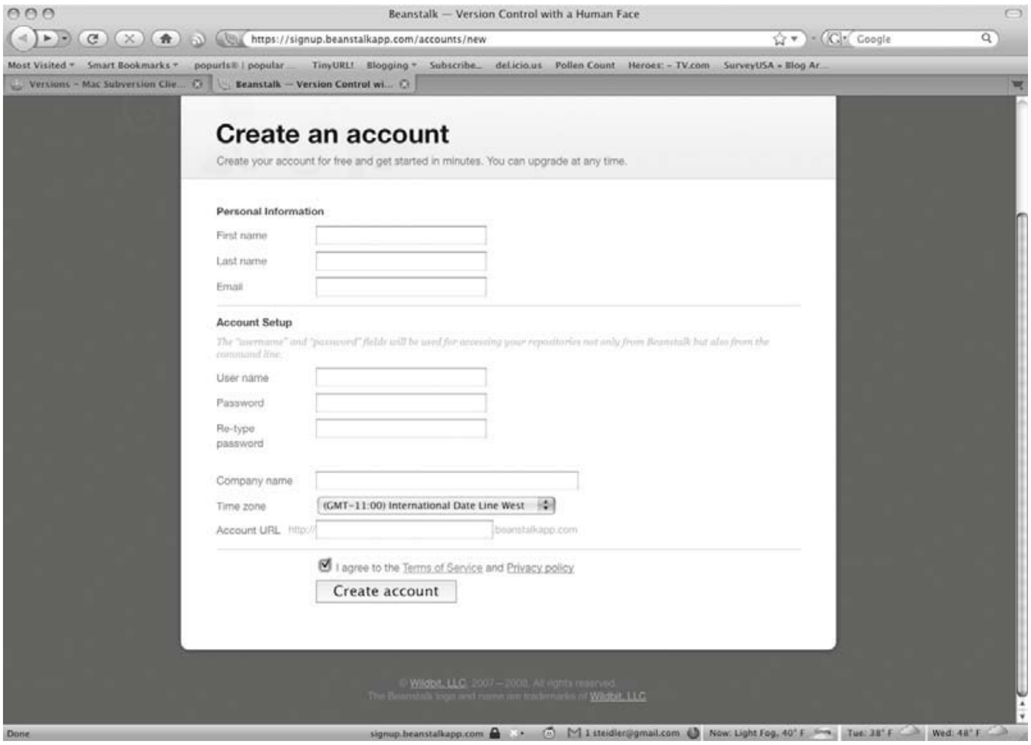
OSXLG_Chap_5_DRAFT.rev176.txt – /var/folders/i5/i5GiAa1c

```
*** NeoOffice
NeoOffice is a fully GPL'ed, OS X productivity suite, based
**** NewOffice Write
*** neooffice_writer_main.png ***
NeoOffice Write is a feature-complete word processing applic
*** neooffice_calc_main.png ***
NeoOffice Calc., like Write, is a fully functional spreadshe
*** neooffice_impress_main.png ***
NeoOffice Impress presents the familiar interface of a power
*** neooffice_draw_main.png ***
While presented as a drawing application, Draw is more analo
*** neooffice_base_main.png ***
NeoOffice Base is a standalone database application, intende
In most significant ways, NeoOffice is an outstanding and po
*** OpenOffice
OpenOffice is a productivity suite originally created as Sta
*** openoffice_main.png ***
OpenOffice, too, contains several tools widely accepted as p
In June 2008, OpenOffice.org released it's first self-conta
* Wrapping Up the Apple and Linux Tools
OS X has many strengths. It's a powerful and versatile platf
```

status: 9 differences

Figure 8-34. *Viewing differences between file commits in Versions*

One of the features of Versions that clearly distinguishes it from other front-end Subversion tools is clear the first time you open the client. The main interface, as shown in Figure 8-35, provides the option to create a free online repository via Beanstalk.


Beanstalk — Version Control with a Human Face

https://signup.beanstalkapp.com/accounts/new

Create an account

Create your account for free and get started in minutes. You can upgrade at any time.

Personal Information

First name

Last name

Email

Account Setup

The "username" and "password" fields will be used for accessing your repositories not only from Beanstalk but also from the command line.

User name

Password

Re-type password

Company name

Time zone: (GMT-11:00) International Date Line West

Account URL http:// .beanstalkapp.com

[✓] I agree to the Terms of Service and Privacy policy

Create account

© Wildbit, LLC 2007 – 2008. All rights reserved.
The Beanstalk logo and name are trademarks of Wildbit, LLC

Figure 8-35. *Setting up a free online repository with Versions*

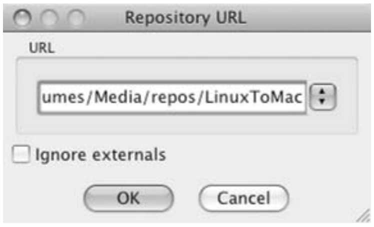
Setting up an online repository is a pretty painless process, requiring only that you establish an online account with Beanstalk, and follow the simple instructions provided once the account is created. Taking advantage of the Beanstalk account provides an additional measure of code security, since the files are stored off-site. Committing, checking out, and viewing files in a Beanstalk repository with Versions is no different from taking those same actions on locally stored files. The drawbacks to the free Beanstalk accounts are the limits of 20MB storage and three developers.

Versions provides access to the full set of Subversion subcommands and options. With the additional bonus of an online Beanstalk repository, it's a tool you'll certainly want to consider when looking at GUI front ends for your Subversion installation.

RapidSVN

Another GUI Subversion tool for Mac OS X is RapidSVN. It's available in `.dmg` image form at `http://rapidsvn.tigris.org/`.

Like Versions, RapidSVN uses bookmarks to create a new repository or to connect to an existing repository. As shown in Figure 8-36, creating a bookmark to a repository in RapidSVN requires only that you configure the URL for the repository. This can be an online repository or a local version.

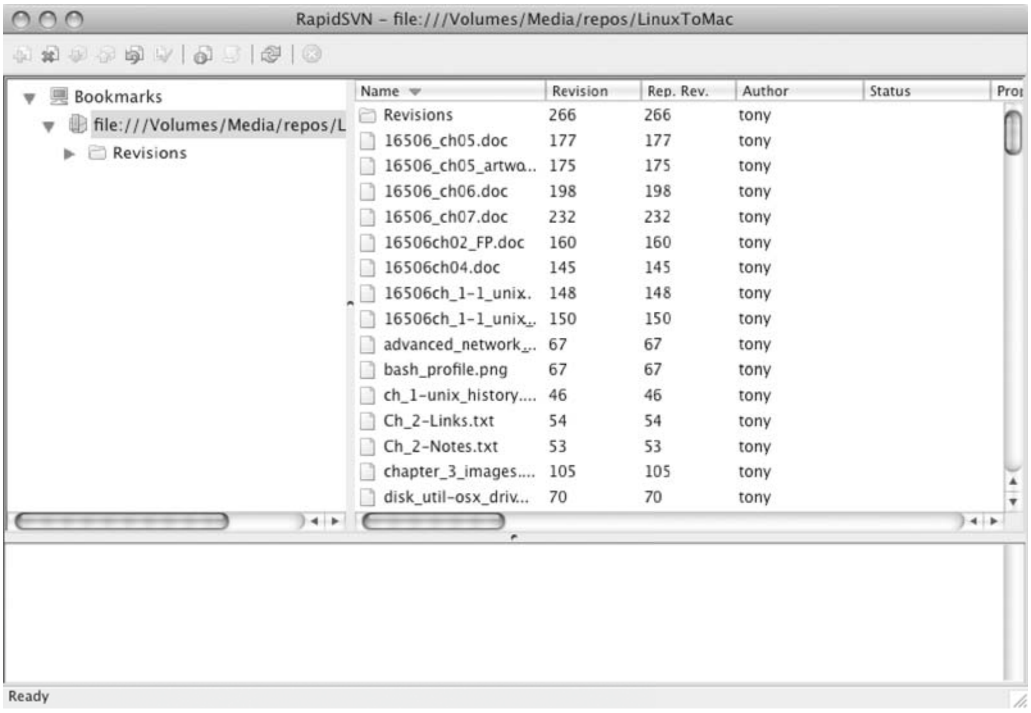


URL: umes/Media/repos/LinuxToMac

☐ Ignore externals

Figure 8-36. *Creating a repository bookmark in RapidSVN*

As with other GUI Subversion tools, the repository is browsable in a single RapidSVN window, as shown in Figure 8-37.


RapidSVN – file:///Volumes/Media/repos/LinuxToMac

- Bookmarks
  - file:///Volumes/Media/repos/L
    - Revisions

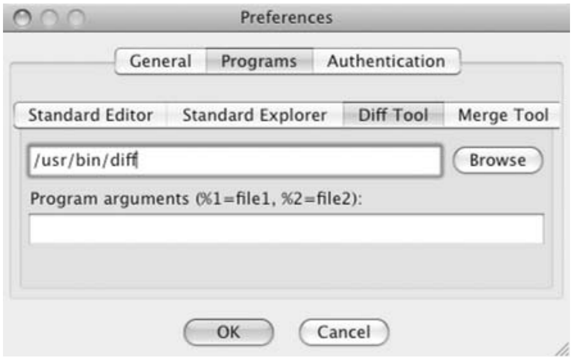
Name	Revision	Rep. Rev.	Author	Status	Pro
Revisions	266	266	tony		
16506_ch05.doc	177	177	tony		
16506_ch05_artwo...	175	175	tony		
16506_ch06.doc	198	198	tony		
16506_ch07.doc	232	232	tony		
16506ch02_FP.doc	160	160	tony		
16506ch04.doc	145	145	tony		
16506ch_1-1_unix.	148	148	tony		
16506ch_1-1_unix...	150	150	tony		
advanced_network...	67	67	tony		
bash_profile.png	67	67	tony		
ch_1-unix_history....	46	46	tony		
Ch_2-Links.txt	54	54	tony		
Ch_2-Notes.txt	53	53	tony		
chapter_3_images....	105	105	tony		
disk_util-osx_driv...	70	70	tony		

Ready

Figure 8-37. *Browsing the repository in RapidSVN*

The RapidSVN Preferences window allows you to configure the various tools used to take actions on the files controlled by Subversion. As shown in Figure 8-38, these include the Diff tool, the Merge tool, and the editor. The Preferences window also provides general configuration options and options for authentication.

Note As is the case in Linux, you can check whether a specific tool is in the path on your Mac OS X system. To do so, simply open a Console window and enter the command `which [tool]`. If the tool is in the path, its location will be returned in the command line. To find the Diff and Merge tools for RapidSVN, for example, I executed the `which` command, which showed that both tools were located in `/usr/bin`.



/usr/bin/diff

Program arguments (%1=file1, %2=file2):

Figure 8-38. *Configuring the RapidSVN preferences*

Like Versions, RapidSVN provides the full set of Subversion tools for version control. However, one of the primary differences between the Versions and RapidSVN applications is the location of these tools. Versions places the most commonly used tools on the toolbar in the main window. RapidSVN places those tools in a context menu launched by right-clicking a file.

Managing Changes with Git

Subversion has the current buzz, but it's not the only version control system for Mac OS X. Git is a scalable, distributed version control system for Linux that installs easily in Mac OS X.

While Git is a powerful version control tool, its real power is in a distributed environment serving many developers. Within the open source community, Git is used in projects as diverse as Linux kernel development, Wine, and X.Org. Like Subversion, Git uses the copy-modify-merge versioning model, allowing users to create local copies of files, and then managing concurrent changes at the time of check-in to the repository. Other Git features include the following:

Git protocol: This is an efficient network protocol created specifically for Git. Users can also optionally use the HTTP protocol to check out and commit files.

Scalability: Git is designed with large projects in mind. It scales quickly and easily to accommodate ever-growing project demands.

UNIX tool approach: Git makes full use of the UNIX philosophy of many small tools that do one thing right. As a collection of these tools written in C, Git provides nearly unlimited flexibility for developers.

Cryptographic history authentication: Git histories are stored in a way that prevents those histories from being changed. This ensures the integrity of the file version histories.

The source code for Git is available at `http://git.or.cz/`. A Mac OS X `.dmg` image is available at `http://code.google.com/p/git-osx-installer/downloads/list?can=3`, as is the `OpenInGitGUI` front-end zip file.

To build and install Git from the source code package, execute the following commands:

```
$ tar zxvf git-1.6.0.4.tar.gz
$ cd git-1.6.0.4
$ ./configure
$ make && sudo make install
```

To install from the `.dmg` image, double-click the image file to mount the image, double-click the installer package, and follow the prompts.

Like Subversion, the command set for Git is deep. The command syntax is as follows:

```
git [--version] [--exec-path[=GIT_EXEC_PATH]] [-p|--paginate|--no-pager] [--bare]
[--git-dir=GIT_DIR] [--work-tree=GIT_WORK_TREE] [--help] COMMAND [ARGS]
```

The most common Git commands are listed in Table 8-8.

Table 8-8. *Common Git Commands*

Command	Description
`add`	Adds file contents to the index
`bisect`	Finds the change that introduced a bug by binary search
`branch`	Lists, creates, or deletes branches
`checkout`	Checks out a branch or paths to the working tree
`clone`	Clones a repository into a new directory
`commit`	Records changes to the repository
`diff`	Shows changes between commits, commit and working tree, and so on
`fetch`	Downloads objects and references from another repository
`grep`	Prints lines matching a pattern
`init`	Creates an empty Git repository or reinitializes an existing one
`log`	Shows commit logs
`merge`	Joins two or more development histories together
`mv`	Moves or renames a file, a directory, or a symlink
`pull`	Fetches from and merges with another repository or a local branch
`push`	Updates remote references along with associated objects
`rebase`	Forward-ports local commits to the updated upstream head
`reset`	Resets the current HEAD to the specified state

Continued

Table 8-8. *Continued*

Command	Description
`rm`	Removes files from the working tree and from the index
`show`	Shows various types of objects
`status`	Shows the working tree status
`tag`	Creates, lists, deletes, or verifies a tag object signed with GnuPG

Summary

Right out of the box, Mac OS X provides a complete environment for developers. It starts with the inclusion of several programming languages, focused both on object-oriented programming and on scripting. From C to Objective-C to Perl, Python, and Java—developers will find those language choices to be nearly complete.

Mac OS X also provides a rich development and debugging environment in Xcode. It's powerful and flexible, with all the features developers have come to expect in a modern IDE. And, as the tool used in the development of Mac OS X itself, Xcode is put to the test every day by Apple.

Finally, Mac OS X provides several great options for source control. While Mac OS X developers may install nearly any open source revision control tool, the system includes Subversion, currently one of the most popular code control tools. And if you're looking to make full use of Subversion without the learning curve required for the command-line tool, several GUI options are available. Versions and RapidSVN are two of those options for GUI-based source and code control with Subversion.

In short, if you've cut your programming teeth in the open source world, you'll find a lot of familiar ground in Mac OS X, and most of the tools provided require no further modification of your system.