AppleScript Studio Programming Guide

Scripting & Automation > AppleScript

2006-04-04

Apple Inc.
© 2001, 2006 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 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 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 computers.

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

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

Apple, the Apple logo, AppleScript, AppleScript Studio, Aqua, Carbon, Cocoa, Mac, Mac OS, Macintosh, Objective-C, QuickTime, and Xcode are trademarks of Apple Inc., registered in the United States and other countries.

Finder is a trademark of Apple Inc.

Java and all Java-based trademarks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

SPEC is a registered trademark of the Standard Performance Evaluation Corporation (SPEC).

UNIX is a registered trademark of The Open Group

Simultaneously published in the United States and Canada.

Even though Apple has reviewed this document, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, 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 DOCUMENT, 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 AppleScript Studio Programming Guide 13

Who Should Read This Document 14
Organization of This Document 14
Conventions 15
See Also 15

Chapter 1 About AppleScript Studio 17

What Is AppleScript Studio? 17
What Makes AppleScript Studio Special? 17
- AppleScript 18
- Integrated Development Environment 18
- Application Framework 19
- Strengths and Limitations 20

How AppleScript Studio Works 21
- AppleScript Studio's Components 21
- AppleScript Studio Applications 22
- Connecting Actions to Scripts 22
- Putting It All Together 25

Creating a Hello World Application 27
AppleScript Studio Sample Applications 35

Chapter 2 AppleScript Studio Components 39

AppleScript Overview 39
- How AppleScript Is Implemented 40
- Scripting in AppleScript Studio 41
- Terms for Classes and Objects 41

Xcode Features for AppleScript Studio 42
- AppleScript Studio Application Templates 43
- AppleScript Studio Xcode Plug-in Template 43
- Default Project Contents 44
- The Targets Group 48
- Source Code Editor 49
- Debugging Features 51
- Terminology Browser 51

Interface Builder Features for AppleScript Studio 53

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Interface Creation 54
Interface Connections 56
Cocoa Framework Overview 58
Cocoa Scripting Support 58
Cocoa User Interface Objects 59
Cocoa Application Framework 59
AppleScriptKit Framework Overview 60

Chapter 3 Programming With AppleScript Studio 61

Additional Information on AppleScript Studio 61
Organizing an AppleScript Studio Project 62
Naming Conventions for Methods and Handlers 64
Accessing Code From AppleScript Studio Scripts 64
Persistent Script Properties 67
Accessing Script Globals 68
Overridden Scripting Additions 68
How Xcode Formats Scripts 69
Switching Between AppleScript Studio and Script Editor 70
Scripting AppleScript Studio Applications 71
Using Script Editor to Test AppleScript Studio Terminology 72
AppleScript Studio Terminology 73
Overview 73
General Sources of Scripting Terminology 74
Terminology From the AppleScriptKit Framework 75
Finding Terminology Information 77
Programming Tips 80
Targeting an AppleScript Studio Application 80
Using Make, Not Create, to Create New Objects in Scripts 80
Using the Log Command to Track Your Scripts 80
Basic Tips and Reminders 81
Troubleshooting 82
My Script Statements Aren't Working 82
Several Windows in My Application Have ID 0 83
I Can't Script My UI to Do QA Testing 83

Chapter 4 AppleScript Studio Cookbook 85

Performing User Interface Actions 85
Specifying Minimum Requirements for an Application 86
Adding AppleScript Studio Support to Your Cocoa Application 86
Setting the Keyboard Focus 87
Obtaining the Path to the Current Application 87

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Chapter 5 Currency Converter Tutorial 89

Design the Application 90
Create a Project 90
Build the Interface 91
Launch Interface Builder 92
Adjust the Title, Size, and Other Attributes of the Currency Converter Window 92
Add Text Input Fields and Labels 98
Add a Result Field and Label 105
Add Number Formatters to the Input and Result Fields 107
Add a Convert Button 110
Add a Horizontal Separator 111
Finalize the Layout 112
Connect the Interface 112
Write Event Handlers 115
Build and Run the Application 117
Where To Go From Here 117

Chapter 6 Mail Search Tutorial: Design the Application 119

Before You Start This Tutorial 120
Identify a Goal for the Application 120
Examine Mail's Scripting Dictionary 120
Specify Operations for Mail Search 122
Design the Interface 123
Identify Objects for the User Interface 123
Arrange the User Interface 125
Plan the Code 127
Event Handlers in Mail Search 128
Additional Handlers and Scripts in Mail Search 129

Chapter 7 Mail Search Tutorial: Create the Interface 133

Create a Project 133
Add an Image File to the Project 135
Build the Interface 136
Examine the Default Menus 136
Create the Message Window 137
Create a Status Dialog 145
Create the Search Window 152

Chapter 8 Mail Search Tutorial: Connect the Interface 169

Connect the Interface 169
Connect the Application Object 169
Connect Interface Items in the Search Window 172

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Chapter 9 Mail Search Tutorial: Write the Code 187

Obtaining the Code for the Mail Search Tutorial 187
Define Global Variables and Properties 188
Write Event Handlers for the Interface 188
- Application Object Handler 189
- Search Window Handlers 189
- Text Field Handler 191
- Find Button Handler 191
- Search Results Table View Handler 192

Write Scripts and Additional Handlers 192
- Write the Controller Script 192
- Write Handlers for Working With Controllers 203
- Write Handlers for Working With the Status Dialog 204
- Write Handlers for Working With Message Windows 204
- Write Utility Handlers 204

Chapter 10 Mail Search Tutorial: Build and Test the Application 205

Build and Run Mail Search 205
Check for Syntax Errors 206

Chapter 11 Mail Search Tutorial: Customize the Application 209

Customize Menus 209
- Rename Menus and Menu Items 209
- Set Menu Attributes 211
- Remove Menus and Menu Items 212

Customize the About Window 212
Customize Version and Copyright Information 214
Customize Icons 215
- Add an Icon Resource File to the Project 217
- Supply a Creator Code 220

Appendix A AppleScript Studio System Requirements and Version Information 221

Appendix B Mail Search Tutorial, Full Script Listing 223

Mail Search Copyright Notice 233

Glossary 235

Document Revision History 239

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Figures, Tables, and Listings

Chapter 1 About AppleScript Studio 17

Figure 1-1 The Drawer sample application 18
Figure 1-2 The components of AppleScript Studio 21
Figure 1-3 Connections between user interface items and scripts in an AppleScript Studio application 23
Figure 1-4 Interface Builder's Palette window, showing the Cocoa-Controls palette 24
Figure 1-5 A window containing a button 24
Figure 1-6 The Info window for a button 25
Figure 1-7 Creating an application with AppleScript Studio 26
Figure 1-8 Choosing an AppleScript Studio application in Xcode 27
Figure 1-9 A newly-created AppleScript Studio application project 28
Figure 1-10 A project with many groups and files expanded 29
Figure 1-11 A default application window in Interface Builder 30
Figure 1-12 Interface Builder's Palette window, showing the Cocoa-Controls palette 30
Figure 1-13 A window containing a button 31
Figure 1-14 The Hello World button 31
Figure 1-15 The Info window for the Hello World button 32
Figure 1-16 Editing a `clicked` handler in Xcode 33
Figure 1-17 The Hello World application in action 34
Listing 1-1 A simple event handler 23

Chapter 2 AppleScript Studio Components 39

Figure 2-1 Default contents of an AppleScript Application project 44
Figure 2-2 Default contents of an AppleScript Document-based Application project 47
Figure 2-3 The Mail Search Target inspector 49
Figure 2-4 Editing a Hello World script in Xcode 50
Figure 2-5 The Open Dictionary dialog in Xcode 51
Figure 2-6 The AppleScript Studio scripting dictionary in a browser window 52
Figure 2-7 Interface Builder windows after opening the `MainMenu.nib` file 54
Figure 2-8 The AppleScript palette in Interface Builder's Palette window 56
Figure 2-9 The Info window for a button 57
Listing 2-1 Full Objective-C code for a simple AppleScript Studio application 59

Chapter 3 Programming With AppleScript Studio 61

Figure 3-1 Syntax for the call method command 65

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Figure 3-2 Setting Text Editing preferences in Xcode 70
Figure 3-3 The AppleScript Studio scripting dictionary in Xcode 78
Table 3-1 Naming conventions in Cocoa and AppleScript Studio 64
Table 3-2 Cocoa types and their AppleScript equivalents 67
Table 3-3 AppleScript Studio sample applications and the objects they use 78
Listing 3-1 Detecting which button was clicked 62
Listing 3-2 Calling a document method with two parameters 66
Listing 3-3 Calling a method of a button 66
Listing 3-4 Calling a class method 66
Listing 3-5 Calling a method of the application 66
Listing 3-6 Telling Script Editor where to look for AppleScript Studio terminology 71
Listing 3-7 Setting text in the Drawer application from an external script 71
Listing 3-8 Scripting the Drawer application from Script Editor 72
Listing 3-9 Examining the views of an AppleScript Studio application 73
Listing 3-10 A new clicked handler 81

Chapter 4 AppleScript Studio Cookbook 85

Listing 4-1 Manipulating a button in the Drawer application from an external script 85
Listing 4-2 will finish launching handler that checks for required version of AppleScript Studio 86

Chapter 5 Currency Converter Tutorial 89

Figure 5-1 The Currency Converter window 90
Figure 5-2 The project after opening several groups 91
Figure 5-3 Selected text for Window instance in MainMenu.nib window 92
Figure 5-4 The default window in the Currency Converter project 93
Figure 5-5 The Attributes pane of the Info window for a window object 94
Figure 5-6 The Info window after retitling the Currency Converter window 95
Figure 5-7 The final Attributes settings for the Currency Converter window 96
Figure 5-8 The Size pane of the Currency Converter window 97
Figure 5-9 The modified Currency Converter window 98
Figure 5-10 The Cocoa-Text palette of Interface Builder's Palette window 99
Figure 5-11 Positioning a text input field for the exchange rate 99
Figure 5-12 Resizing the exchange rate input field 100
Figure 5-13 The Attributes pane of the Info window for the exchange rate field 100
Figure 5-14 The Info window, after supplying an AppleScript name for the exchange rate field 101
Figure 5-15 The MainMenu.nib window showing an AppleScript Info object (not selected) 102
Figure 5-16 Positioning a label field for the exchange rate 102
Figure 5-17 Resizing the label field for the exchange rate 103
Figure 5-18 The Info window, after setting text and attributes for the exchange rate label 104
Figure 5-19 The exchange rate label field (selected) 104

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Figure 5-20 The Currency Converter window with input fields and labels 105
Figure 5-21 The Currency Converter window with all text fields and labels 106
Figure 5-22 The Info window, after disabling editing for the amount in other currency field 107
Figure 5-23 Adding a number formatter to the exchange rate input field 108
Figure 5-24 The Formatter pane for the exchange rate input field 109
Figure 5-25 The Formatter pane for the amount in other currency field 110
Figure 5-26 The Currency Converter window with a "Convert" button 111
Figure 5-27 The Currency Converter window with a horizontal separator 111
Figure 5-28 Fields from the Info window for setting keystroke equivalent 112
Figure 5-29 Fields from the Info window for setting keystroke equivalent 112
Figure 5-30 The Info window after connecting a clicked handler to the Convert button 113
Figure 5-31 The Info window after connecting a handler to the application object 115
Figure 5-32 The Currency Converter application in action 117
Listing 5-1 The empty clicked handler 115
Listing 5-2 The complete clicked handler 116
Listing 5-3 The should quit after last widow closed handler 117

Chapter 6 Mail Search Tutorial: Design the Application 119

Figure 6-1 The Mail application's scripting dictionary in an Xcode window 121
Figure 6-2 Mail Search's search window in Interface Builder 125
Figure 6-3 A status dialog in Interface Builder 126
Figure 6-4 Mail Search's menu nib in Interface Builder, showing the application menu 126
Figure 6-5 A Mail Search message window in Interface Builder 127

Chapter 7 Mail Search Tutorial: Create the Interface 133

Figure 7-1 Default contents of a document-based AppleScript Studio project 134
Figure 7-2 Deleting a file from a project 134
Figure 7-3 Adding a file to a project 135
Figure 7-4 Interface Builder windows after opening Mail Search's `MainMenu.nib` file 137
Figure 7-5 Creating a new nib file in Interface Builder 138
Figure 7-6 A new nib file in Interface Builder 139
Figure 7-7 The Cocoa-Windows palette of Interface Builder's Palette window 140
Figure 7-8 The Message.nib window, showing a window instance 140
Figure 7-9 The AppleScript pane in the Info window for a window object 141
Figure 7-10 The Message.nib window showing an AppleScript Info object (not selected) 142
Figure 7-11 The Cocoa-Text palette of Interface Builder's Palette window 143
Figure 7-12 Positioning a text view object 143
Figure 7-13 The finished message window 144
Figure 7-14 Attributes pane in Info window for text vie 145

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Figure 7-15 The status dialog as previously designed 146
Figure 7-16 The revised Attributes pane in the Info window for the status dialog 147
Figure 7-17 The Size pane in the Info window for the status dialog 147
Figure 7-18 The resized status dialog 148
Figure 7-19 The Cocoa-Controls palette of Interface Builder's Palette window 148
Figure 7-20 Positioning the progress bar 149
Figure 7-21 Resizing the progress bar 149
Figure 7-22 Positioning a status text field above the progress bar 150
Figure 7-23 Resizing the status text field 150
Figure 7-24 The Attributes pane in the Info window for the text field 151
Figure 7-25 The invisible status text field 151
Figure 7-26 The resized, empty search window 152
Figure 7-27 Positioning a popup button in the search window 153
Figure 7-28 The default contents of a popup button 154
Figure 7-29 A popup button with a new item 154
Figure 7-30 A renamed popup button item 154
Figure 7-31 Popup button with renamed items 155
Figure 7-32 The popup button after checking the Small checkbox 155
Figure 7-33 Resizing the popup button 155
Figure 7-34 Positioning a text field in the search window 156
Figure 7-35 Resized text field in the search window 156
Figure 7-36 Positioning a button in the search window 157
Figure 7-37 The Info window for the find button 157
Figure 7-38 The button and magnifying glass 158
Figure 7-39 Aligning the right edge of the window with the button 158
Figure 7-40 Inserting an outline view in the search window 159
Figure 7-41 Resizing the outline view in the search window 160
Figure 7-42 A selected outline view 161
Figure 7-43 The Info window for the outline view, after changes 162
Figure 7-44 The outline view after naming the Mailboxes column 162
Figure 7-45 The Info window for the scroll view containing the outline view 163
Figure 7-46 Inserting a table view in the search window 164
Figure 7-47 The Info window for the table view, after changes 165
Figure 7-48 The table view with titles 166
Figure 7-49 The final search window, now containing a split view 167

Chapter 8 Mail Search Tutorial: Connect the Interface 169

Figure 8-1 The File's Owner instance in the MainMenu.nib window 170
Figure 8-2 The Info window for the File's Owner instance 171
Figure 8-3 The Info window for the Mail Search window instance 173
Figure 8-4 The Info window for the search text field 175
Figure 8-5 The Info window for the find button 176
Figure 8-6 The Info window for the search results table view 178
Figure 8-7 The Document.nib window with a data source object 179
Figure 8-8 Connecting the outline view to the data source object 180

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Figure 8-9 The Info window for the outline view after connecting a data source outlet 181
Figure 8-10 The Info window after entering an outline column identifier 182
Figure 8-11 The Document.nib window 183
Figure 8-12 The Document.nib window in outline view 183
Figure 8-13 Connections for the NSOutlineView object 184
Figure 8-14 The Info window for the table view after connecting a data source outlet 185
Figure 8-15 The Info window after entering a table column identifier 186
Listing 8-1 A new handler declaration for the `will finish launching` handler 172
Listing 8-2 New handler declarations for several handlers 174
Listing 8-3 A new `action` handler for a text field 175

Chapter 9 Mail Search Tutorial: Write the Code 187

Listing 9-1 Global list variable to store instances of controller script 188
Listing 9-2 Properties used in Mail Search 188
Listing 9-3 The `will finish launching` handler for the application object 189
Listing 9-4 `will open` handler for search window 190
Listing 9-5 The `became main` handler for the search window 190
Listing 9-6 The `will close` handler for the search window 190
Listing 9-7 The `action` handler for the search text field 191
Listing 9-8 The `clicked` handler for the find button 191
Listing 9-9 The `double clicked` handler for the search results table view 192
Listing 9-10 Properties of the controller script 193
Listing 9-11 The controller script's `initialize` handler 193
Listing 9-12 The controller script's `loadMailboxes` handler 194
Listing 9-13 The controller script's `addMailboxes` handler 195
Listing 9-14 The controller script's `addAccounts` handler 195
Listing 9-15 The controller script's `addMailbox` handler 196
Listing 9-16 The controller script's `find` handler 197
Listing 9-17 The controller script's `findMessages` handler 199
Listing 9-18 The controller script's `openMessageWindow` handler 201
Listing 9-19 The `makeMessageWindow` handler 204
Listing 9-20 Utility function to delete an item from a list 204

Chapter 10 Mail Search Tutorial: Build and Test the Application 205

Figure 10-1 An uncompiled handler 206
Figure 10-2 A syntax error in an Xcode script editor window 207
Figure 10-3 A compiled handler 208
Figure 10-4 Event handlers in an Xcode pop-up menu 208

Chapter 11 Mail Search Tutorial: Customize the Application 209

Figure 11-1 Mail Search's menu nib in Interface Builder, showing the application menu 210

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Figure 11-2 The revised Mail Search application menu 210
Figure 11-3 The Info window for the About Mail Search menu item 211
Figure 11-4 AppleScript Studio's default About window 213
Figure 11-5 The About window after modifying the application description 214
Figure 11-6 The About window after modifying version and copyright information 215
Figure 11-7 Mail Search's icons displayed in Icon Composer 216
Figure 11-8 Adding a file to a project 217
Figure 11-9 The Icon field in a target window for the Mail Search target 218
Figure 11-10 The About window after customizing icons 219
Figure 11-11 The Mail Search icon in the Finder 220
Listing 11-1 The default InfoPlist.strings file from an AppleScript Studio application 215

Appendix A AppleScript Studio System Requirements and Version Information 221

Table A-1 Availability for AppleScript Studio development environment and runtime 221

Appendix B Mail Search Tutorial, Full Script Listing 223

Listing B-1 Mail Search's global variables and event handlers 223
Listing B-2 The controller script definition 224
Listing B-3 Handlers for working with controller script objects 231
Listing B-4 The message window handler 232
Listing B-5 The status dialog script definition 232
Listing B-6 The delete items in list utility 233

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Introduction to AppleScript Studio Programming Guide

Note: This document was previously titled "Building Applications With AppleScript Studio."

Important: This is a preliminary draft of AppleScript Studio documentation. Although it has been reviewed for technical accuracy, some information is subject to change. In particular, information about AppleScript Studio features that became available after version 1.1 is incomplete. Screenshots and tutorial instructions for tools such as Xcode and Interface Builder refer to Mac OS X version 10.3 versions of the tools.

For the latest documentation on AppleScript Studio, see *AppleScript Studio Terminology Reference*.

AppleScript Studio Programming Guide provides the key information you'll need to create AppleScript Studio applications.

AppleScript Studio is a powerful tool for quickly creating native Mac OS X applications that support the Aqua user interface guidelines. It combines features from AppleScript, Xcode, Interface Builder, and the Cocoa application framework. With AppleScript Studio, you can work in a full-featured development environment to create applications that use AppleScript scripts to control a broad range of Cocoa user-interface objects.

Note: AppleScript Studio requires Mac OS X version 10.1.2 or later, both to build and to deploy applications. See "Appendix A, AppleScript Studio System Requirements and Version Information," (page 221) for more information.

AppleScript Studio has something to offer both to scripters and to those with Cocoa development experience:

- It provides access to AppleScript's ability to control multiple applications, including parts of the Mac OS itself.
- Scripters can create applications with a complex user interface, including windows, buttons, menus, text fields, tables, and much more. Scripts have full access to user interface objects.
- Cocoa developers can use AppleScript Studio to speed up prototyping, testing, and deploying of applications.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Who Should Read This Document

This document assumes that you have some familiarity with AppleScript and know how to write and execute scripts.

Previous experience building applications with an integrated development environment is also recommended—familiarity with Xcode and Interface Builder is especially useful.

Previous experience with Cocoa is not required, but can be helpful in understanding some of AppleScript Studio's underlying mechanisms.

For documentation and other resources for these technologies, see "See Also" (page 15).

For information on whether AppleScript Studio is appropriate to your task, see "Strengths and Limitations" (page 20).

Organization of This Document

This document contains the following chapters:

- "Introduction to AppleScript Studio Programming Guide" (page 13) briefly describes AppleScript Studio, provides a description for each chapter, and lists some related documentation.
- "About AppleScript Studio" (page 17) introduces AppleScript Studio's key features and shows how to create a simple "Hello World" application.
- "AppleScript Studio Components" (page 39) provides a detailed description of AppleScript Studio, including descriptions of the key features in Xcode and Interface Builder, as well as overviews of the Cocoa and AppleScriptKit frameworks.
- "Programming With AppleScript Studio" (page 61) describes additional features and issues you'll want to know more about as you work with AppleScript Studio. It also describes the scripting terminology you need to write scripts and provides tips for programming with AppleScript Studio.
- "AppleScript Studio Cookbook" (page 85) provides step-by-step instructions for performing some common AppleScript Studio tasks.
- "Currency Converter Tutorial" (page 89) provides a simple tutorial that introduces the tools and processes you'll use in most AppleScript Studio development.
- "Mail Search Tutorial: Design the Application" (page 119) is the first of several chapters that make up a tutorial for a more complex AppleScript Studio application. This chapter describes the process of designing the application.
- "Mail Search Tutorial: Create the Interface" (page 133) describes how to create the interface for the Mail Search application.
- "Mail Search Tutorial: Connect the Interface" (page 169) shows how to connect Mail Search's interface to event handlers in the application's scripts.
- "Mail Search Tutorial: Write the Code" (page 187) describes the handlers and script statements for the Mail Search application.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

- "Mail Search Tutorial: Build and Test the Application" (page 205) provides information on how to build and test the Mail Search application.
- "Mail Search Tutorial: Customize the Application" (page 209) provides steps for customizing menus, icons, and version and copyright information in the Mail Search application.
- "AppleScript Studio System Requirements and Version Information" (page 221) describes the system requirements for building and running AppleScript Studio applications.
- "Mail Search Tutorial, Full Script Listing" (page 223) contains a complete listing of the Mail Search application's script file.
- "Document Revision History" (page 239) describes changes made to this document.
- "Glossary" (page 235) defines key terms for working with AppleScript Studio.

Conventions

You'll see the AppleScript continuation character (¬, which you create by typing Option-l) in some of the script listings in this document. When a line in a script ends with a continuation character, the next line is considered to be part of that line. You shouldn't need the continuation character when you actually compile the scripts in AppleScript Studio, because you can use Xcode's ability to wrap text instead. For more information, see "How Xcode Formats Scripts" (page 69).

Some listings in this document may use wrapped text, rather than the continuation character.

See Also

You can find getting started and overview documentation for AppleScript, AppleScript Studio, and related technologies, with links to all the available Apple documentation and resources (including mailing lists), here:

- Getting Started With AppleScript
- *AppleScript Overview*

Because AppleScript Studio relies heavily on the Cocoa application framework, you may also want to visit the Cocoa Documentation area, particularly these documents:

- *Application Architecture Overview*
- *Cocoa Scripting Guide*

You can also use any web search engine to many third-party books, products, and websites for AppleScript and AppleScript Studio.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

About AppleScript Studio

AppleScript Studio is a powerful tool for quickly creating native Mac OS X applications that support the Aqua user interface guidelines. AppleScript Studio applications use AppleScript scripts to control complex user interfaces. This chapter introduces AppleScript Studio, provides a basic description of its key features, and shows you how to build a simple application.

Note: AppleScript Studio requires Mac OS X version 10.1.2 or later, both to build and to deploy applications. See "Appendix A, AppleScript Studio System Requirements and Version Information," (page 221) for more information.

What Is AppleScript Studio?

AppleScript Studio is a combination of application framework and development environment. It combines features from AppleScript, Xcode, Interface Builder, and the Cocoa application framework. Together, these components provide a sophisticated environment for creating AppleScript solutions. Using AppleScript Studio:

- Scripters can build native Mac OS X applications that execute AppleScript scripts, have access to a wide range of user interface objects, and can control scriptable applications and scriptable parts of the Mac OS. These applications are referred to as **AppleScript Studio applications**.
- Cocoa developers can take advantage of AppleScript's many features, including controlling other applications, and can add sophisticated scripting capabilities (not currently available in Cocoa alone) to their applications.

What Makes AppleScript Studio Special?

AppleScript Studio is special because it makes it easier to create Mac OS X applications with complex user interfaces that can communicate with and control other applications. The following sections describe additional features that help make AppleScript Studio one of a kind.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

AppleScript

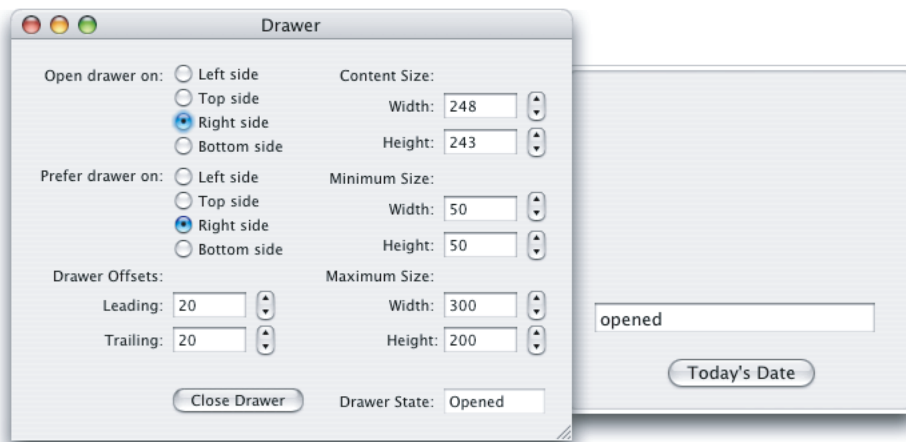
AppleScript provides the powerful ability to control multiple applications, including many parts of the Mac OS itself. That allows scripters to set up workflow solutions with a combined power that exceeds that of any individual application.

In addition to the ability to control multiple applications, AppleScript's strengths include:

- an English-like language that makes scripts easier to write and understand
- powerful language features, including list and record manipulation, as well as script objects that provide features such as inheritance and encapsulation; script objects are described in "Additional Handlers and Scripts in Mail Search" (page 129)
- the ability to target applications on remote machines
- with the addition of support for **SOAP** (simple object access protocol) and **XML-RPC** (a simple protocol for making remote procedure requests to Internet-based servers) in Mac OS X version 10.1, the ability to target Internet servers with remote procedure calls

With AppleScript Studio, script developers can take advantage of all these features, while quickly creating applications that include complex user interfaces. For example, Figure 1-1 shows Drawer, one of many sample applications distributed with AppleScript Studio. Drawer demonstrates how to use a number of interface classes, including buttons, text fields, radio buttons, steppers, and even its namesake, a drawer. Other sample applications display file and folder information in windows similar to the Finder's column and list view. See "AppleScript Studio Sample Applications" (page 35) for a complete list of sample applications.

Figure 1-1 The Drawer sample application



Integrated Development Environment

Because it is integrated with Apple's development environment, AppleScript Studio can take advantage of powerful features provided by Xcode and Interface Builder. These include:

- use of Cocoa's rich set of user interface classes; layout tools include built-in support for Aqua interface guidelines

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

- tools that simplify building and maintaining complex projects with multiple targets and build steps
- easy customization of application menus, icons, and About windows

AppleScript Studio supports a number of scripting features that are also available in the Script Editor application distributed with Mac OS X version 10.3, but were not available in previous versions of the Script Editor. These include:

- creation of arbitrarily large scripts
- search and replace in scripts
- easy access to handlers and properties in scripts (through a pop-up menu)
- a flexible dictionary viewer for working with application scripting terminologies

For more information, see "Xcode Features for AppleScript Studio" (page 42) and "Interface Builder Features for AppleScript Studio" (page 53).

Application Framework

Because AppleScript Studio applications are Cocoa applications, they benefit from Cocoa's full-featured application framework. As a result, an AppleScript Studio application can perform many operations automatically, without any additional Objective-C code from the developer. Built-in features allow users to open multiple windows, resize and minimize windows, display an About window, enter text in text fields, and even shuffle column positions in a table view.

> **Note:** Objective-C is Cocoa's native programming language, but you can use other kinds of code within an AppleScript Studio application. For more information, see the section "Accessing Code From AppleScript Studio Scripts" (page 64) and the description of the Multi-Language application in "AppleScript Studio Sample Applications" (page 35).

To experiment with the features you get in the simplest document-based AppleScript Studio application, even before adding any code or scripts, see the steps in "Create a Project" (page 133).

Users with previous Cocoa experience will also find a lot to like in AppleScript Studio, including the ability to

- use AppleScript to control other applications
- do quick prototyping, with scripts taking the place of unimplemented methods
- perform simple automated testing, using AppleScript Studio's ability to script Cocoa user interface objects (not available in Cocoa alone); for more information, see "Scripting AppleScript Studio Applications" (page 71)

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Note: AppleScript Studio also supports calling code directly from scripts. For more information, see "Accessing Code From AppleScript Studio Scripts" (page 64).

Although you can create applications that perform virtually all of their operations by executing AppleScript scripts, you are free to include additional Cocoa code in applications. You may find it useful to learn more about the Cocoa code working behind the scenes—to do so, see the information provided in "Cocoa Framework Overview" (page 58), as well as the Cocoa documentation described in "See Also" (page 15).

Strengths and Limitations

AppleScript Studio offers a number of powerful features. However, for scripting tasks that don't require a complex user interface, such as adjusting your workspace or automating repetitive tasks, the Script Editor (distributed with the Mac OS) or a third-party scripting application is usually a more appropriate tool. You'll have access to AppleScript's key features without the overhead that comes with AppleScript Studio's additional power.

AppleScript Studio shows its strength for tasks that require:

- a complex user interface
- manipulation of information associated with user interface elements
- display of information provided or manipulated by other processes (including information gathered from databases)
- the ability to take advantage of features written in standard programming languages (which you can access from AppleScript Studio scripts, as shown in the Multi-Language sample application, described in "AppleScript Studio Sample Applications" (page 35))
- a fully functional build environment

AppleScript Studio is less appropriate for tasks that require:

- display of large amounts of data (such as massive tables)
- intensive computation or manipulation of large amounts of data in AppleScript Studio scripts
- intensive interaction with a file system (such as displaying large numbers of files; you can try the Browser sample application, described in "AppleScript Studio Sample Applications" (page 35), to experiment with the performance of an application that browses the file system)
- simple scripting operations, especially those with little or no user interface

You may notice that AppleScript Studio performs poorly when you use AppleScript scripts to perform computation-intensive operations. This reflects the limits of the processing power of the AppleScript language, which was not designed for those kinds of tasks. One way to work around these issues is to have your scripts call into C, C++, Objective-C, or Java code to perform computation-intensive operations. The Multi-Language sample application, distributed with AppleScript Studio, demonstrates how to access code written in various languages from an AppleScript Studio application.

AppleScript Studio does not support building non-Cocoa applications, or applications that must run in Mac OS 9, or in versions of Mac OS X before version 10.1.2.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

AppleScript Studio applications, like other Cocoa applications, can access frameworks and libraries outside the Cocoa framework, including the Carbon framework, although detailed steps for doing so are not described in this document.

How AppleScript Studio Works

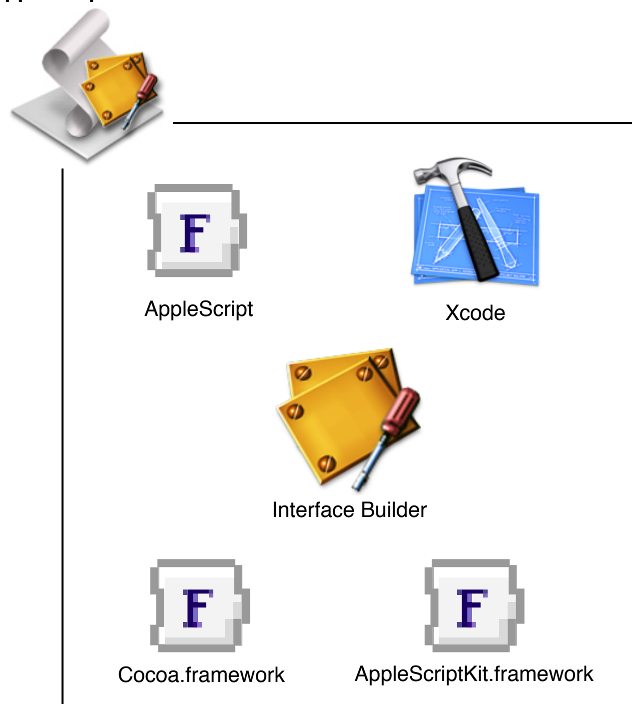
This section provides a brief description of AppleScript Studio's components, introduces key concepts, and lists the steps required to create an AppleScript Studio application. It contains the following

"AppleScript Studio's Components" (page 21)
"AppleScript Studio Applications" (page 22)
"Connecting Actions to Scripts" (page 22)
"Putting It All Together" (page 25)

AppleScript Studio's Components

Figure 1-2 shows the components that make up AppleScript Studio: AppleScript, Xcode, Interface Builder, the Cocoa framework, and AppleScript Studio's own framework, the AppleScriptKit framework. A **framework** is a type of bundle (or directory in the file system) that packages software with the resources that software requires, including its interface.

Figure 1-2 The components of AppleScript Studio



2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Here's how AppleScript Studio's components work together to produce AppleScript Studio applications:

- **AppleScript:** Provides the ability to control multiple applications, including parts of the Mac OS, by writing scripts. For more information, see "AppleScript Overview" (page 39).
- **Cocoa framework:** Provides an application framework, including a robust set of user interface classes. You use these classes in Interface Builder to create an interface for your AppleScript Studio application. You also link with Cocoa in Xcode to build the application itself. For more information, see "Cocoa Framework Overview" (page 58).
- **Interface Builder:** Provides a graphical environment for creating user interface descriptions. You also use Interface Builder to link user actions, such as clicking a button or choosing an item in a pop-up menu, to specific handlers in scripts. (Handlers are described in "Connecting Actions to Scripts" (page 22).) For more information, see "Interface Builder Features for AppleScript Studio" (page 53).
- **Xcode:** Provides the development environment to edit, build, and debug AppleScript Studio applications, as well as to display dictionaries of scripting terms. For more information, see "Xcode Features for AppleScript Studio" (page 42).
- **AppleScriptKit framework:** Provides code and scripting terminology to support AppleScript Studio features, including enhanced scriptability for user interface objects and the ability to call Objective-C methods from scripts. For more information, see "AppleScriptKit Framework Overview" (page 60).

See "Chapter 3, AppleScript Studio Components," (page 39) for a more detailed description of AppleScript Studio's components.

AppleScript Studio Applications

AppleScript Studio applications take advantage of the Cocoa framework, which works "behind the curtain" to display the interface, respond to user actions, and more. However, there is very little visible Cocoa code required for an AppleScript Studio application (see Listing 2-1 (page 59). As a result, scripters gain the ability to create complex interfaces and work in a powerful development environment, while still being able to use AppleScript script statements to control applications.

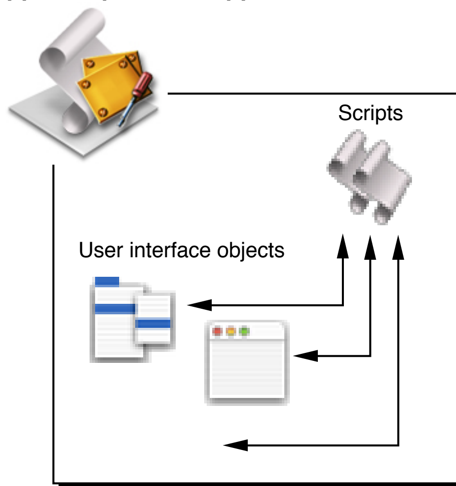
Connecting Actions to Scripts

The ability to connect application events to scripts is a key concept in understanding AppleScript Studio. In every AppleScript Studio application you write, you will connect actions involving the application's user interface objects to event handlers in its script files, the result of which is shown in Figure 1-3.

A **handler** is a series of one or more script statements that are executed in response to an action or condition. In the case of a simple subroutine, a handler is similar to a function. Handlers that respond to action events in the application are called **event handlers** to distinguish them from other handlers.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Figure 1-3 Connections between user interface items and scripts in an AppleScript Studio application



Handlers can have zero or more parameters. Listing 1-1 shows a simple event handler, where `(*Add your script here.*)` is an AppleScript comment. All event handlers start with the keyword `on`. The `clicked` handler has one parameter, `theObject`, which represents the user interface object that received the `clicked` message. Most event handlers in AppleScript Studio have this same parameter.

Listing 1-1 A simple event handler

```
on clicked theObject
    (*Add your script here.*)
end clicked
```

You'll see the full details later in this chapter, but in brief, the process of connecting actions to event handlers consists of these steps:

1. Use Interface Builder to create a new user interface resource file (or nib file), or to open an existing nib file. The default nib file in an AppleScript Studio project, called `MainMenu.nib`, automatically contains one window object.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

2. Add interface objects (such as buttons and text views) in Interface Builder. Figure 1-4 shows one of the palettes of objects available in Interface Builder.

Figure 1-4 Interface Builder's Palette window, showing the Cocoa-Controls palette

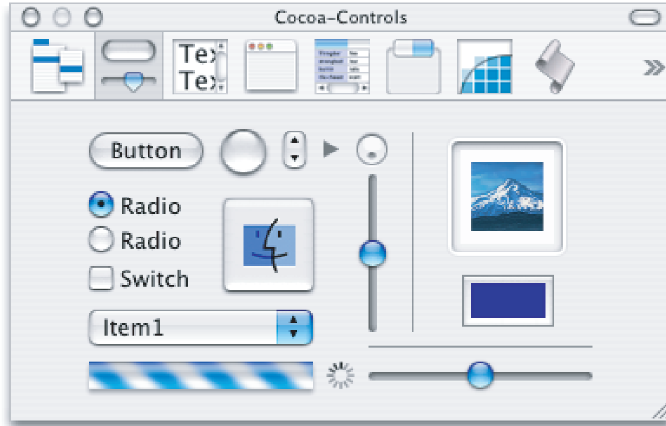


Figure 1-5 shows the results of dragging a button object from the palette to a window.

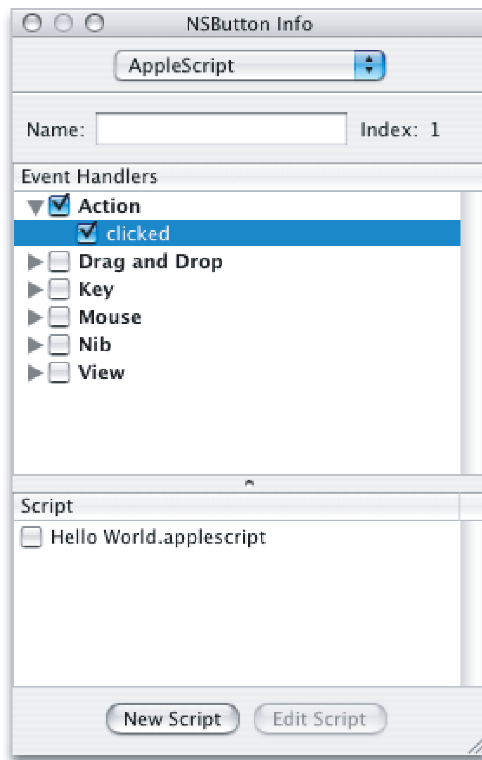
Figure 1-5 A window containing a button

3. Select an interface item (such as a button) in Interface Builder and open the Info window. Figure 1-6 shows the Info window for a button, with the AppleScript pane visible, showing the possible event handler groups for the object. Each group contains one or more handlers.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

4. Click to select the checkbox for the event handler (or handlers) you want to connect, then select a script file to connect it to, as shown in Figure 1-6. The available event handlers are grouped according to their function. The Action group contains just the `clicked` handler. Interface Builder inserts an empty event handler, in this case the `clicked` handler shown in Listing 1-1, into the selected script file in your application project.

Figure 1-6 The Info window for a button



5. Open the script in Xcode and write script statements for the event handler. You can open the script by clicking the Edit Script button.

These steps are shown in more detail in "Creating a Hello World Application" (page 27).

Putting It All Together

Previous sections have described AppleScript Studio's components, AppleScript Studio applications, and the key steps for connecting actions to scripts. By adding just a bit more detail, you have an algorithm for creating an application with AppleScript Studio:

1. Use Xcode to create a new project, using one of the AppleScript Studio application templates Xcode supplies.

2. Build the application's user interface (which is stored in user interface resource files, or nib files) by adding user interface objects with Interface Builder.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

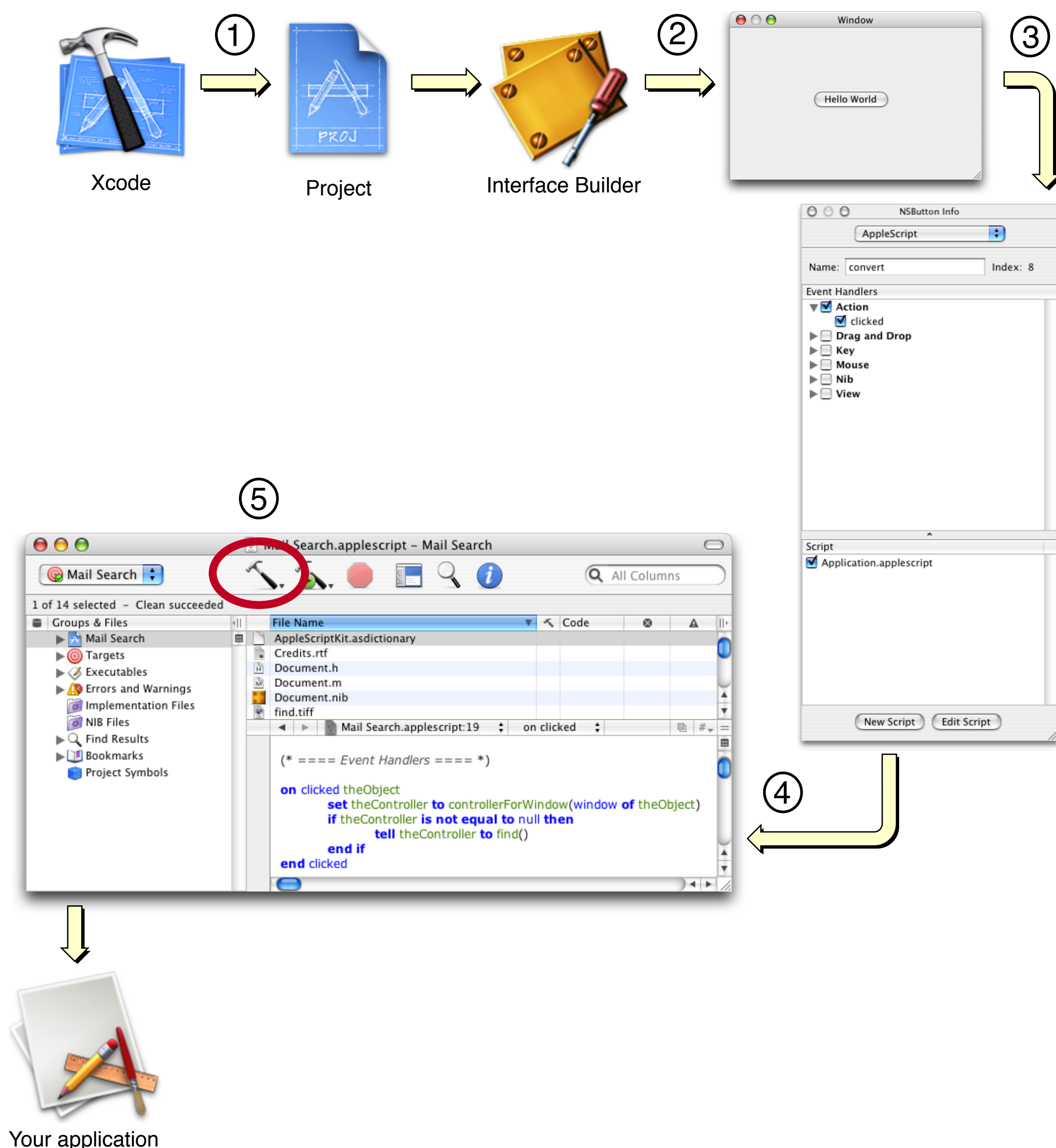
3. Use the Info window in Interface Builder to hook up user actions, such as mouse clicks and menu selections, to event handlers in scripts.

4. Edit the scripts in Xcode to add script statements that perform the desired operations.

5. Build the application in Xcode.

A pictorial version of these steps is shown in Figure 1-7, and a simple tutorial demonstrates them in "Creating a Hello World Application" (page 27). Of course you may need to repeat steps 2 through 5 as you build and test your application. You'll find more information on these operations throughout this document.

Figure 1-7 Creating an application with AppleScript Studio



2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Creating a Hello World Application

This section shows how easy it is to create a simple "Hello World" application with AppleScript Studio. See "Chapter 6, Currency Converter Tutorial," (page 89) for a slightly more complex tutorial that demonstrates additional features of AppleScript Studio. See "Chapter 7, Mail Search Tutorial: Design the Application," (page 119) to start a multi-chapter tutorial that builds a more complete AppleScript Studio application.

If you'd like to know more about AppleScript Studio before you start programming, feel free to read "Chapter 3, AppleScript Studio Components," (page 39) before going ahead with the Hello World tutorial.

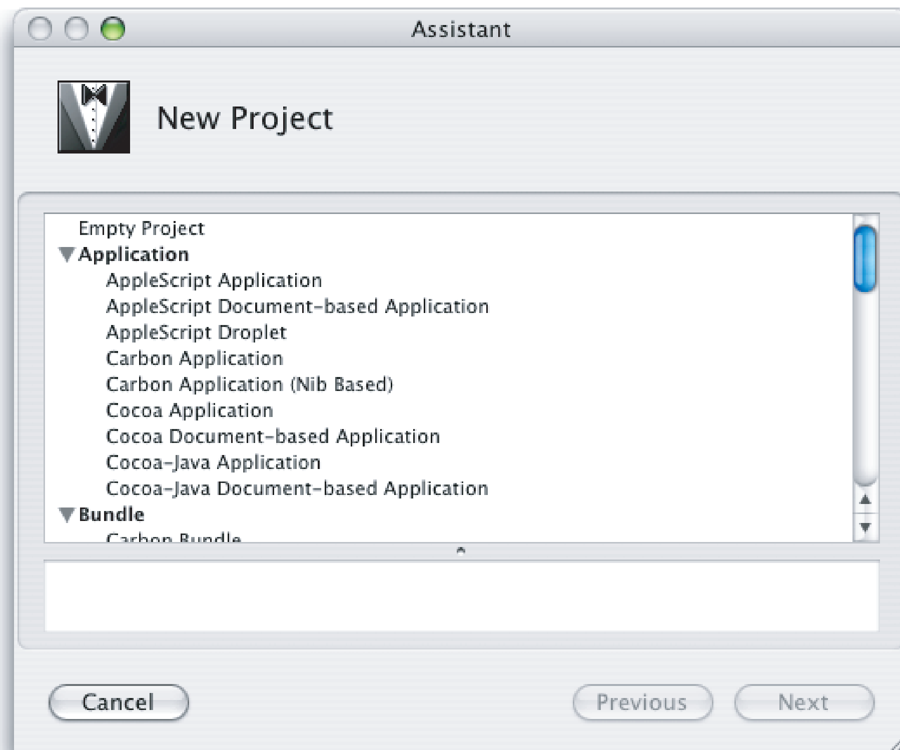
To create a "Hello World" AppleScript Studio application, perform these steps:

1. See the information in "AppleScript Studio System Requirements and Version Information" (page 221) to make sure that AppleScript Studio is available on your computer.

2. Open the Xcode application, located in `/Developer/Applications`.

3. Choose New Project from the File menu. Xcode opens the dialog shown in Figure 1-8:

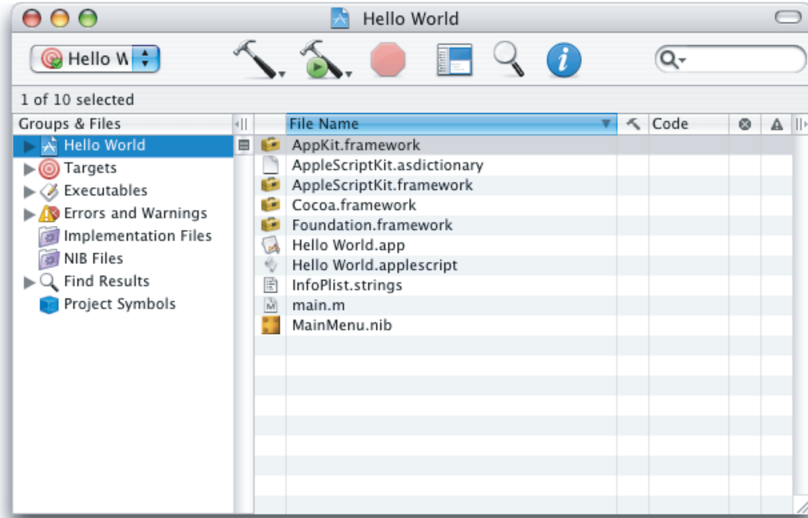
Figure 1-8 Choosing an AppleScript Studio application in Xcode



2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

4. Select one of the AppleScript Studio application templates. For this example, choose the "AppleScript Application" template, then click the Next button. You will then get a chance to name the project and save it in the location of your choice. Type "Hello World" (without the quotes) for the project name. The resulting project is shown in Figure 1-9.

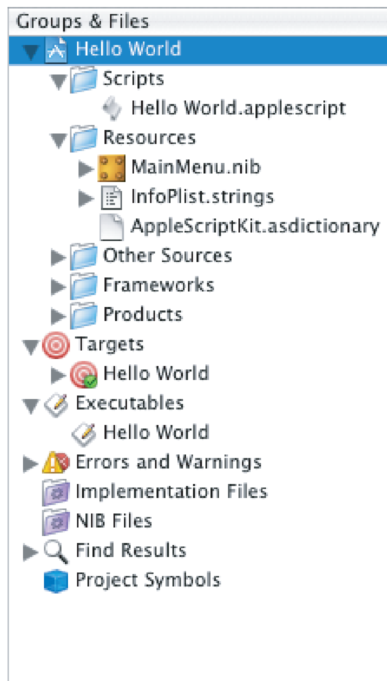
Figure 1-9 A newly-created AppleScript Studio application project



2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

5. The project's contents are visible in the detail view (to the right of the Groups & Files list). You can click the disclosure triangles in the Groups & Files list to see how the contents are stored in the project. Figure 1-10 shows the Groups & Files list with most groups expanded. The items are described in "Default Project Contents" (page 44).

Figure 1-10 A project with many groups and files expanded



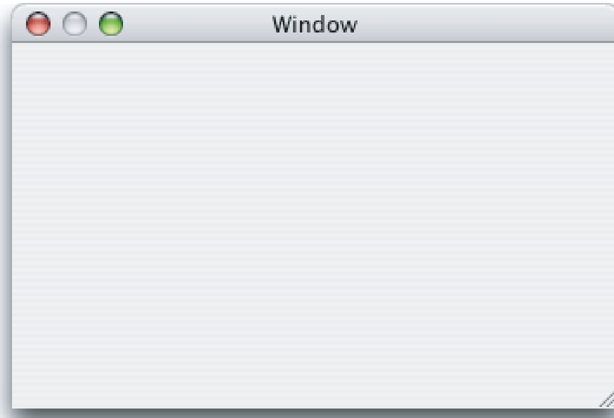
6. Save the new project by typing Command-S or by choosing the Save command from the File menu.

Important: Don't forget to save your project periodically as you make changes to it.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

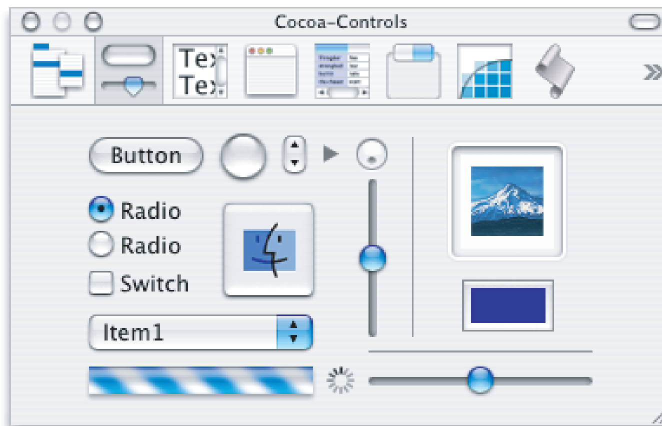
7. Double-click the icon for the `MainMenu.nib` file to open the Interface Builder application (located in `/Developer/Applications`). Interface Builder opens a number of windows. One of those windows, the default application window is shown in Figure 1-11. Drag the resize control in the lower-right corner of the window to reduce the window to a size suitable for containing one button.

Figure 1-11 A default application window in Interface Builder



8. Figure 1-12 shows the Palette window, another window that opens when you open Interface Builder. You can change the current palette by clicking one of the buttons in the window's toolbar.

Figure 1-12 Interface Builder's Palette window, showing the Cocoa-Controls palette



2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Drag a button from the Cocoa-Controls palette into the "Window" window; the result is shown in Figure 1-13.

Figure 1-13 A window containing a button

9. Double-click Button to select the button title text, then type "Hello World" as the button title. The result is shown in Figure 1-14.

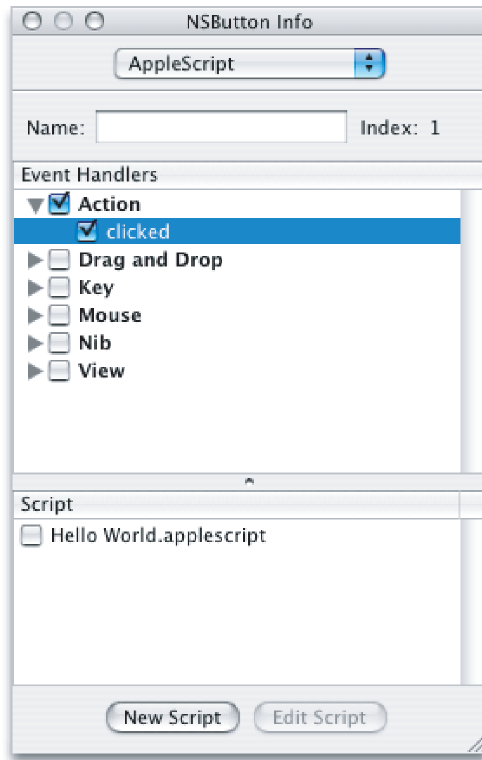
Figure 1-14 The Hello World button

10. With the Hello World button selected, choose Show Info from the Tools menu (or type Command-Shift-I) to open the Info window. Use the pop-up menu at the top of the window to display the AppleScript pane. Then click the disclosure triangle next to the Action checkbox (if it isn't already open) and click the "clicked" checkbox. The result is shown in Figure 1-15.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Selecting the clicked action indicates that the Hello World button should have a `clicked` event handler, to be called when a user clicks the button.

Figure 1-15 The Info window for the Hello World button



11. In the Script pane in the Info window shown in Figure 1-15, select the checkbox for `Hello World.applescript`, the default script for the application. That tells AppleScript Studio to put the `clicked` handler for the Hello World button in the selected script. In this example there is only one script—for an example that uses multiple scripts, see the Display Panel sample application (distributed with AppleScript Studio).

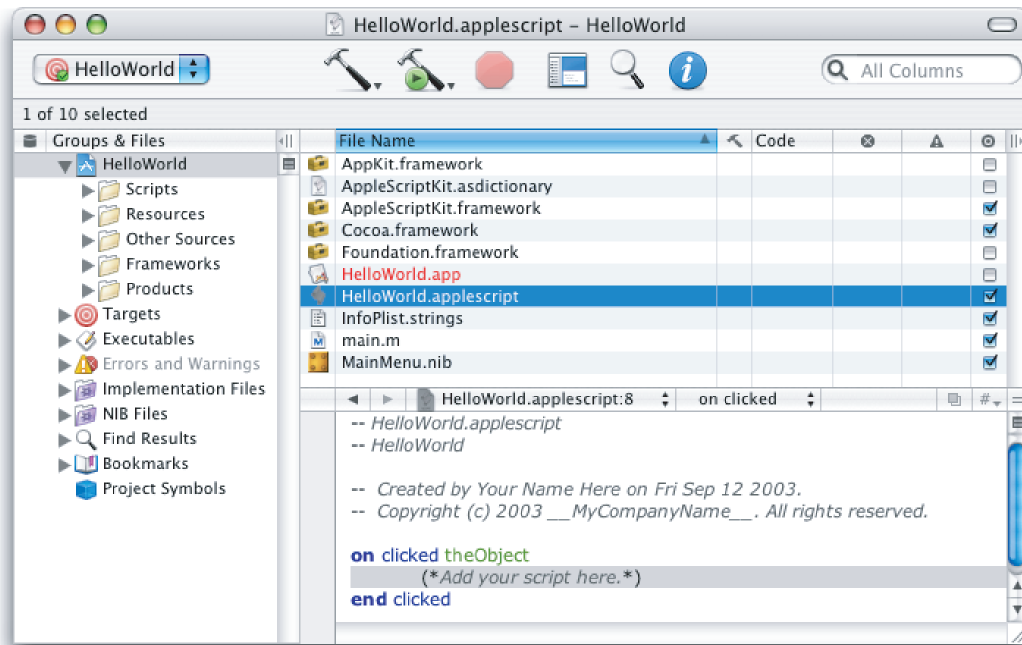
    When you click the Edit Script button, Xcode becomes active to allow you to edit the selected script, as shown in Figure 1-16. AppleScript Studio has already inserted the following handler, which is called whenever a user clicks the button:

    ```
    on clicked theObject
        (*Add your script here.*)
    ```


2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

```
end clicked
```

Figure 1-16 Editing a `clicked` handler in Xcode



12. Select the text `(*Add your script here.*)` and replace it with `display dialog "Hello World!"`

    You could, of course, insert many other kinds of AppleScript statements as part of the handler executed when a user clicks the button.

13. Build the Hello World application by typing Command-B or choosing Build from the Build menu. (You can also use one of the build buttons, which have a hammer icon.)

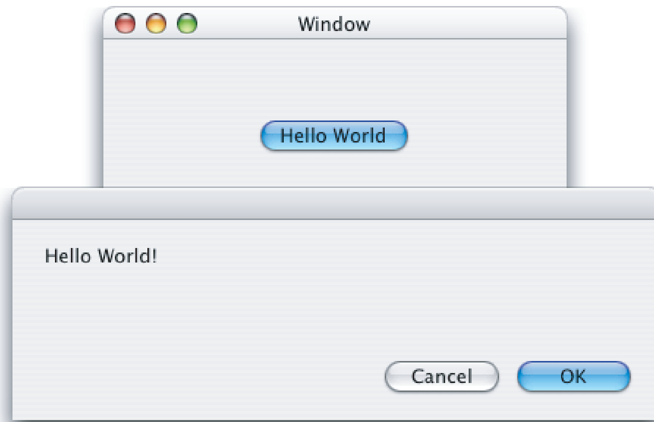
    The script `Application.applescript` is compiled automatically when you build the application. Later chapters in this document describe how to compile scripts separately.

    Building the application also compiles the Cocoa code in the application's `main.m` file, prepares the application's resources, and links the application with the required frameworks.

14. Run the Hello World application by typing Command-R or choosing Build and Run from the Build menu. (Both of these options actually rebuild the application, but since you already built it in the previous step, the application should open quickly.) Figure 1-17 shows the result of clicking the Hello World button.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Figure 1-17 The Hello World application in action



Building this Hello World application required very little Objective-C code, and that code was auto-generated by AppleScript Studio. In fact, you can build quite complex AppleScript Studio applications with no additional Cocoa code. However, every AppleScript Studio application is a Cocoa application, and it's the Cocoa application framework that's working behind the scenes. This suggests several possible paths for Studio developers, depending on your background and goals:

- Anyone interested in AppleScript Studio should:
  - See "Chapter 3, AppleScript Studio Components," (page 39) for a more detailed introduction to AppleScript Studio.
  - Work through the tutorials in this document and experiment with the applications described in "AppleScript Studio Sample Applications" (page 35).
  - See "Terminology From the AppleScriptKit Framework" (page 75) to learn more about the user interface objects you can use with AppleScript Studio and about the classes, properties, events, and enumerations you can use in scripts. (This section in turn points to the document *AppleScript Studio Terminology Reference*, the best source for terminology information.)

- If you have a scripting background and your main goal is to create scripting applications with complex user interfaces:
  - Use your scripting knowledge, along with information in this document and *AppleScript Studio Terminology Reference*, to help you quickly take advantage of AppleScript Studio.
  - Be aware of AppleScript Studio's Cocoa underpinnings, and that the interface objects you use are implemented by Cocoa classes. Don't be afraid to occasionally read the Cocoa information described in this document or pointed to elsewhere.

- If you have a Cocoa background and you'd like to take advantage of features of AppleScript Studio:
  - See "AppleScript Overview" (page 39) for an introduction to AppleScript.
  - Refer to the documents described in "See Also" (page 15) as needed for more information on scripting.
  - Use your knowledge of Cocoa, along with information in this document, to provide a head start in understanding and taking advantage of Cocoa's important role in AppleScript Studio.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

❏ See the document *AppleScript Studio Terminology Reference*, which describes the terms you use to work with buttons, menus, views, and other interface items in AppleScript Studio scripts. Most AppleScript Studio classes are based on Cocoa user interface classes, and where this is the case, the document contains a link to the Cocoa documentation for the corresponding Cocoa class.

AppleScript Studio Sample Applications

AppleScript Studio includes the following sample applications, which can serve as a valuable source of examples and coding techniques for working with the many available user interface objects. The applications are located in `/Developer/Examples/AppleScript Studio/`. Table 3-3 (page 78) lists the main user interface objects used in some of the applications.

Sample applications that were added after AppleScript Studio version 1.1 are so noted. For related information, see "AppleScript Studio System Requirements and Version Information" (page 221).

- **Archive Maker** demonstrates many Cocoa user interface objects, such as drawers and panels, in a graphical front end to the `gnutar` shell tool (for building tar archives). It also shows how to use a `call method` statement to call an Objective-C method, and how to use a `do shell script` statement to execute a shell command.

  AppleScript's `do shell script` command is documented in "Technical Note TN2065: `do shell script` in AppleScript" at http://developer.apple.com/technotes/tn2002/tn2065.html.

- **Assistant** presents one possible implementation of an Assistant, using a tab view with separate tab view items to represent an information panel. The tab view has no border or visible tabs, which supports the appearance of switching a full panel of user interface elements in and out.
- **Browser** browses the file system, displaying files and folders in a window similar to the Finder's column view.
- **Command Finder** finds shell commands that match specified text strings, according to the current search type in a pop-up menu (such as "begins with" or "contains"). Double-clicking a found command opens a window containing the man page for that command.
- **Coordinate System** demonstrates how to specify the coordinate system for an application. Added in AppleScript Studio version 1.4 (along with the support for alternate coordinate systems).
- **Countdown Timer** demonstrates how to use an `idle` event handler to build a countdown timer. It also shows how to display an alert as a sheet. Added in AppleScript Studio version 1.2.1.
- **Currency Converter** is a simple application that converts a dollar amount to an amount in another currency.
- **Currency Converter (SOAP)** is a version of the Currency Converter that uses SOAP commands to look up exchange rates. Added in AppleScript Studio version 1.2.
- **Daily Dilbert** uses the `date` and `curl` shell script commands to load and display an image, given a URL from a web service. Added in AppleScript Studio version 1.2.
- **Debug Test** provides a test bed for debugging an AppleScript Studio script. It displays a progress bar and an incrementing text field so that you can set break points and examine values.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

- **Display Alert** demonstrates the `display alert` command, which you can use in place of `display dialog` to alert the user to some condition. You can specify the alert icon, whether to display the alert as a sheet (a modal dialog attached to a document window), the alert message, and the alert button titles. The window also displays the button returned when the user closes the alert.
- **Display Dialog** demonstrates various ways of using the `display dialog` command. The application provides a window for specifying the text, buttons, and other inputs for a `display dialog` command, including whether to display the dialog as a sheet. The window also displays information returned when the user closes the dialog.
- **Display Panel** demonstrates the `display panel` command, which allows you to create your own dialogs and display them either as a dialog or attached to a window as a sheet. The application shows how to load and display a panel from a nib (or user interface resource) file. Also demonstrates the use of multiple scripts in a project.
- **Drag and Drop** demonstrates how a number of user interface items can accept drags, including files dragged from the Finder into a table, a color dragged from a color well into a table, text dragged into a button to change its title, text dragged into a text field, and images dragged into an image view. Added in AppleScript Studio version 1.2.
- **Drag Race** is an amusing demonstration of racing buttons.
- **Drawer**, shown in Figure 1-1 (page 18), demonstrates how to use a number of interface classes, including buttons, text fields, radio buttons, steppers, and yes, even drawers.
- **Image** demonstrates how to load an image and add it to an image view.
- **Language Translator** is another sample that uses SOAP requests—in this case, to translate text to other languages.
- **Mail Search** (in AppleScript Studio 1.0, Mail Search was known as "Watson") is an application that searches specified Mail application mailboxes for any specified text, either in the Subject or To fields or in the body of messages. It demonstrates how to work with outline and table views, as well as progress bars. The Mail Search tutorial, which begins in "Chapter 7, Mail Search Tutorial: Design the Application," (page 119) provides a detailed tutorial for building Mail Search.
- **Multi-Language** demonstrates how to access code written in various languages from an AppleScript Studio application. It works with C, C++, Objective-C, Objective-C++, and Java (both directly and through the Mac OS X Java bridge).
- **Open Panel** demonstrates how to use the open-panel class, either as a modal panel or as a panel attached to a window. You can access an open panel as a property of the Application class.
- **Outline** uses an outline view to browse the file system, displaying files and folders in a window similar to the Finder's list view.
- **Outline Reorder** demonstrates how to set the contents of an outline view using a single list of records. It also shows how to turn automatic reordering support on or off dynamically. Added in AppleScript Studio version 1.4 (along with the support for automatic reordering in data views).
- **Plain Text Editor** provides a simple example of how to write a document-based plain-text editor. It takes advantage of the lower level handlers for document handling, namely `read from file` and `write to file`, so that it can read text documents created by other applications. (The Task List sample shows how to use the two higher level handlers, `data representation` and `load data representation`.) Added in AppleScript Studio version 1.2.
- **Save Panel** demonstrates how to use the save-panel class, either as a modal panel or as a panel attached to a window. You can access a save panel as a property of the Application class.
- **Simple Outline** provides an example of how to populate an outline view using a data source. Added in AppleScript Studio version 1.2.1.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Updated in AppleScript Studio 1.4 to demonstrate the new drag and drop support for data views and the new `change cell` value event handler added.

- **Simple Shell** shows how to use the `do shell script` command to make UNIX shell calls (such as `ls`, `man`, and so on).
- **Simple Table** provides an example of how to populate an outline view using a data source. Added in AppleScript Studio version 1.2.1.

  Updated in AppleScript Studio 1.4 to demonstrate how to set the contents of a table view using a list of lists of strings, as well as how to use the new drag and drop event handlers for data views.
- **Simple Toolbar** demonstrates how to create a toolbar in a window and populate it with toolbar items. Added in AppleScript Studio version 1.4.
- **SOAP Talk** is a tool that helps to build the syntax needed to make SOAP requests over the Internet.
- **Table** demonstrates two ways to work with table views (data displayed in rows and columns): with a data source object to supply table data (the approach also used by the Mail Search sample application), and without a data source object (supplying the data directly from a script). Data source objects are described in "Interface Creation" (page 54).
- **Table Reorder** demonstrates how to set the contents of a table view using a single list of records. It also shows how to turn automatic reordering support on or off dynamically. Added in AppleScript Studio version 1.4 (along with the support for automatic reordering in data views).
- **Table Sort** demonstrates how to add sorting to a table. Clicking in a column header sorts the table based on that column. Clicking more than once in the same column header changes the sort order of that column. Added in AppleScript Studio version 1.2.
- **Talking Head** shows how to use a movie view to display QuickTime movies.
- **Task List** demonstrate how to write a document-based application using the higher level handlers `data representation` and `load data representation`. These handlers allow you to return and set any type of data, but the documents you create with them are readable only by your application. (The Plain Text Editor sample shows how to use the lower level handlers `read from file` and `write to file`.) Task List also demonstrates how to work with a table view, including support for sorting, as well as with menu items. Added in AppleScript Studio version 1.2.
- **Unit Converter** converts values between different units of measurement.
- **XMethods Service Finder** demonstrates how to use Web Services. It uses a service from `XMethods.org` that provides information about all of the services available at that site. It also shows how to create and use a data sources with a table view. Added in AppleScript Studio version 1.2.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

AppleScript Studio Components

AppleScript Studio is a combination of a development environment and an application framework that lets you quickly build AppleScript Studio applications—Mac OS X applications that combine AppleScript scripts and Cocoa user-interface objects. It also supplies enhanced scripting capabilities to Cocoa applications.

AppleScript Studio makes use of features from AppleScript, Xcode, Interface Builder, and the Cocoa application framework. See "About AppleScript Studio" (page 17) for a brief introduction to AppleScript Studio.

This chapter describes, in the following sections, the key features you use to create AppleScript Studio applications:

- "AppleScript Overview" (page 39) describes the scripting system you use in AppleScript Studio.
- "Xcode Features for AppleScript Studio" (page 42) describes AppleScript Studio–specific features in this integrated development environment.
- "Interface Builder Features for AppleScript Studio" (page 53) describes features for creating user interfaces for AppleScript Studio applications.
- "Cocoa Framework Overview" (page 58) describes the sophisticated application framework you use to create AppleScript Studio applications.
- "AppleScriptKit Framework Overview" (page 60) describes the framework that allows AppleScript Studio applications to script Cocoa user interface objects.

For additional overview information, see the chapter "Terminology Fundamentals" in *AppleScript Studio Terminology Reference*.

AppleScript Overview

AppleScript is a scripting system that provides direct control of scriptable Macintosh applications, including many parts of the Mac OS itself. Instead of using a mouse or keyboard to manipulate applications, you create scripts—sets of English-like instructions—to automate tasks, control applications on local or remote computers, and even control complex workflows. There are examples of AppleScript scripts throughout this document.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

AppleScript is the scripting language of choice for many Macintosh users. To provide scripters with increased flexibility and power, developers make applications scriptable, or capable of responding to Apple events. Although Carbon and Cocoa applications use different internal mechanisms to support scripting, scripters can access features in any scriptable application, allowing them to combine features from many applications.

Sophisticated scripters use AppleScript to provide customized solutions that typically make use of multiple scriptable applications, as well as scriptable parts of the Mac OS, including:

- The Finder, which supports a host of operations on files, folders, windows, and other components of the Macintosh desktop
- Mail, which supports operations such as getting, reading, and deleting mail. AppleScript Studio includes the Mail Search sample application, described in detail starting in "Chapter 7, Mail Search Tutorial: Design the Application," (page 119) which uses scripts to search mailboxes in the Mac OS X Mail application.
- QuickTime Player, which supports operations such as opening, playing, and stepping through movies. The Talking Head sample application shows how to play QuickTime movies in an AppleScript Studio application.

AppleScript scripts can target applications on remote machines and can even make remote procedure calls to access services over the Internet (starting with Mac OS X version 10.1). AppleScript's support for remote procedure calls, including XML-RPC and SOAP requests, provides access to a variety of web-based servers that can check spelling, translate text between languages, obtain stock quotes, and more. The SOAP Talk AppleScript Studio application, described in "AppleScript Studio Sample Applications" (page 35), shows one way to provide a user interface to access SOAP services. For documentation on AppleScript's support for remote procedure calls, see *XML-RPC and SOAP Programming Guide*, available in AppleScript Documentation.

As an established scripting language, supported by a large number of scriptable applications and available on almost every Mac, AppleScript is the driving force behind AppleScript Studio.

How AppleScript Is Implemented

You can use AppleScript Studio to create complex scripting solutions without detailed knowledge of how AppleScript is implemented in Mac OS X. Therefore, this section provides only a brief description of that implementation. For a more detailed description, see the additional documentation listed in "See Also" (page 15).

The following are some of the key frameworks and other components that support scripting in Mac OS X. Note, however, that you aren't required to work directly with any of these frameworks in your AppleScript Studio project.

- The **Open Scripting Architecture (OSA)** provides an API for compiling and executing scripts, and for creating scripting components, such as AppleScript itself. It is implemented in `OpenScripting.framework`, a subframework of `Carbon.framework`.
- The **Apple Event Manager** provides an API for sending and receiving Apple events and working with the information they contain. Scripting components use Apple events to send commands and data to targeted processes. The Apple Event Manager is implemented in `AE.framework`, a subframework of `ApplicationServices.framework`.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

- The **AppleScript component** in Mac OS X implements the AppleScript scripting language. A scripting component provides services for compiling and executing scripts (and relies on the OSA).
- The **Script Editor application** provides the ability to edit, compile, and execute scripts, to examine the scripting terminology of scriptable applications (the terms you can use in scripts to communicate with applications), and to perform other scripting tasks. When you use it to execute a script, Script Editor calls on the AppleScript component to convert script statements into the appropriate Apple events to the targeted applications. In Mac OS X, Script Editor is located in `/Applications/AppleScript`. Third-party script editors provide many additional features.

  Most scripters are familiar with Script Editor, and you can learn more about it in AppleScript Help in the Help Center. The version released with Mac OS X version 10.3 provides a number of new and enhanced features, such as a pop-up menu for inserting common script coding blocks, and histories of previous results and event logs.

Scripting in AppleScript Studio

An AppleScript script can target any scriptable application in the Mac OS, including those in the Classic environment and on remote machines. It can also target scriptable parts of the Mac OS, and use any scripting features provided by AppleScript. With few exceptions, you can use any statements in an AppleScript Studio script that you would use in any other AppleScript script in Mac OS X.

AppleScript Studio also supports a key additional feature: the ability to access Cocoa's rich set of user interface objects in scripts. These objects range from buttons and text fields to pop-up menus, browsers, and table views. For more information, see "Cocoa User Interface Objects" (page 59)

To write scripts for an AppleScript Studio application, you need to know what terminology you can use with the user interface objects in the application. For information on how to find the available terminology, see "AppleScript Studio Terminology" (page 73) in this document, as well as *AppleScript Studio Terminology Reference*. In addition, Cocoa's user interface objects are described in "Cocoa User Interface Objects" (page 59).

Terms for Classes and Objects

This document uses the following definitions to help distinguish between various kinds of Cocoa and AppleScript classes and objects you work with in AppleScript Studio:

- A **Cocoa user interface class** is an Objective-C class, generally defined in the AppKit framework, that supports a user interface item. For example, NSButton and NSBrowser are Cocoa user interface classes.
- A **Cocoa user interface object** is an instance of a Cocoa user interface class. When creating the interface for an AppleScript Studio application with Interface Builder, for example, you drag user interface objects such as buttons and browsers into your application windows, where the objects are instances of NSButton and NSBrowser, respectively.

  When you select a Cocoa user interface object in Interface Builder and examine it in the Info window, you see its class name in the window's title bar (for example, NSButton).
- An **AppleScript object class** is a category for AppleScript objects that share characteristics, such as properties and elements. For example, if you open the dictionary for an AppleScript Studio application, you see `button` in the list of classes in the Control View suite.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

- An **AppleScript object** is an instance of an AppleScript object class. An AppleScript object is typically a distinct object in an application or its documents that can be specified in a script. In an AppleScript Studio application, for example, you write statements similar to the following:

  ```
  set currentButton to button "Search" of window "Some Window"
  ```

  where `currentButton` is a reference to an AppleScript object specified by `button "Search"`. The button referred to is also a Cocoa user interface object (an instance of NSButton) in the application. For user interface objects in AppleScript Studio applications, there is usually a corresponding Cocoa object for each AppleScript object.

Xcode Features for AppleScript Studio

Xcode is Apple's integrated development environment for Mac OS X. It supports building Cocoa and Carbon applications (as well as bundles, frameworks, plug-ins, and tools) with C, C++, Objective-C, and Java. Xcode has extensive online help and release notes. There are also several tutorials available that describe how to build and debug standard applications.

When AppleScript Studio is installed, Xcode provides support for building AppleScript Studio applications. See "Appendix A, AppleScript Studio System Requirements and Version Information," (page 221) for information on how to make sure AppleScript Studio is available.

You use the following Xcode features to create AppleScript Studio applications:

- **Project templates** Xcode provides project templates for three types of AppleScript Studio application: AppleScript Application, AppleScript Document-based Application, and AppleScript Droplet. These templates are described in "AppleScript Studio Application Templates" (page 43) and "Default Project Contents" (page 44).

  Starting in AppleScript Studio version 1.3, Xcode also provides a template you can use to add features to Xcode itself. This template is described in "AppleScript Studio Xcode Plug-in Template" (page 43).

- **Interface support** AppleScript Studio relies on Interface Builder to create a user interface for AppleScript Studio applications, as described in "Interface Builder Features for AppleScript Studio" (page 53). As you work on an AppleScript Studio application, you can easily switch back and forth between Xcode and Interface Builder.

- **Source code editor** Xcode's source code editor allows you to edit and compile AppleScript scripts. Scripts can be arbitrarily long and you can perform search and replace and other standard operations. For more information, see "Source Code Editor" (page 49).

- **Terminology Browser** Xcode provides an Open Dictionary command (in the File menu) to view the terminology for any scriptable application. For details, see "Terminology Browser" (page 51).

- **Debugging support** Xcode provides AppleScript Studio with debugging support that allows you to perform a variety of debugging tasks. However, that support is not currently complete. For more information, see "Debugging Features" (page 51).

- **Xcode Documentation Window** Xcode's Help menu provides the Documentation Window for quick access to reference, release notes, websites, and other documentation for working with AppleScript Studio.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

The following sections describe these features in more detail. You can also find step-by-step instructions for working with Xcode in the Mail Search tutorial, starting in "Chapter 7, Mail Search Tutorial: Design the Application," (page 119) and in "Creating a Hello World Application" (page 27).

AppleScript Studio Application Templates

To create a new AppleScript Studio application project in Xcode, you choose New Project from the File menu. The resulting template window is shown in Figure 1-8 (page 27). Xcode provides three kinds of templates for AppleScript Studio projects: AppleScript Application, AppleScript Document-based Application, and AppleScript Droplet. You use the first for applications that don't need documents, the second for applications that create and manage multiple documents, and the third for simple droplets (defined in "AppleScript Droplet Template" (page 47)).

When you create a new project, you specify a name, which becomes the default name for the application as well. You can also choose a location for the project. The default location is your home directory (~/), or the last location you specified when creating a previous project.

AppleScript Studio Xcode Plug-in Template

Starting in AppleScript Studio version 1.3, first distributed with Mac OS X version 10.3, Xcode provides a new template for creating an AppleScript plug-in for Xcode. That is, you can use AppleScript Studio to create a plug-in that adds features to Xcode itself.

To create an AppleScript plug-in project in Xcode, you follow these steps:

1. Choose File > New Project

2. In the New Project Assistant window, scroll down to the section for Standard Apple Plug-ins.

3. Select AppleScript Xcode Plugin

Once you've built your plug-in, you place it in one of the following locations so that it will be loaded by Xcode the next time Xcode is launched:

- `~/Library/Application Support/Apple/Developer Tools/Plug-Ins`

  check for plug-ins in the user's home directory

- `/Library/Application Support/Apple/Developer Tools/Plug-Ins`

  check in the system domain

- `/Network/Library/Application Support/Apple/Developer Tools/Plug-Ins`

  check in the network domain

For an example of a plug-in script, see the Example section for the `plugin loaded` event in the PlugIn suite in *AppleScript Studio Terminology Reference*.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Default Project Contents

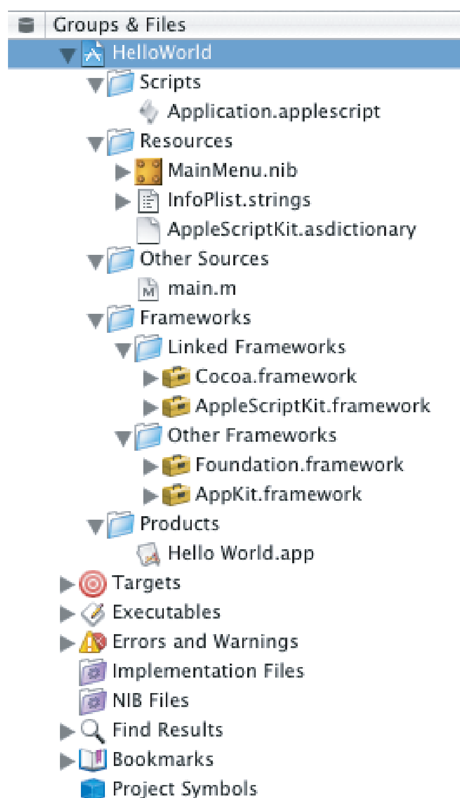
Xcode organizes a project's files in a nested hierarchy of groups, displayed in the Files list in the project's Groups & Files list, as shown in Figure 2-1. A group is represented by a folder icon, but items in a group do not necessarily reside in the same folder. If an item has a checked checkbox in the Target column (the column headed by a target symbol, next to the Groups & Files column), it is part of the current target and is included when that target is built. You can add an item to a build or remove it by clicking to select or deselect its checkbox in the target column. For more information on targets and builds, see the Documentation Window in the Xcode Help menu.

The default contents of the Groups & Files list is different for a new project based on the AppleScript Application and AppleScript Droplet templates than for one based on the AppleScript Document-based Application template, as described in the following sections.

AppleScript Application Template

A new project based on the AppleScript Application template contains all of the items shown in Figure 2-1.

Figure 2-1 Default contents of an AppleScript Application project



The following is a description of these default items:

- The Scripts group contains the application's script files, all of which must have the extension `.applescript` so that Xcode's source code editor recognizes them as AppleScript script files.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

`Application.applescript` is the default script file, which is created as an empty file. When you write script statements to perform operations in your application, you do so in this file, or in additional script files you create.

- The Resources group contains all of the application's resources, and you should store any resource you add in this group. The default resources include the following:

  - `AppleScriptKit.asdictionary` is a pointer to a file in the AppleScriptKit framework (described below). This file describes the scripting terminology you can use in AppleScript Studio scripts. It includes terms that are available to all Cocoa applications that support scripting (in the Standard and Text suites) and those that are specific to AppleScript Studio applications (the Application, Container View, Control View, Data View, Menu, Panel, and Text View suites). See "AppleScript Studio Terminology" (page 73) for more information on these suites.

    `AppleScriptKit.asdictionary` is provided so that you can conveniently examine the terminology before the application is built (and without opening the dictionary of some other built AppleScript Studio application). You can click the file's icon to examine its contents; for details, see "Terminology Browser" (page 51).

    Note: Unlike the formatting for script files in an AppleScript Studio project, the formatting in `AppleScriptKit.asdictionary` is fixed when the file is created, and does not change to match settings you specify in the preferences for the Script Editor application. (In versions of Script Editor released prior to Mac OS X version 10.3, you set formatting by choosing the AppleScript Formatting menu item.) However the formatting for dictionaries from other applications that you open will reflect your settings. For information on script formatting, see "How Xcode Formats Scripts" (page 69).

  - `MainMenu.nib` is the main nib file (or user interface resource file) for the application. Nib files are described in "Interface Builder Features for AppleScript Studio" (page 53). You double-click the nib icon to open the file in Interface Builder, where you can add interface items, assign event handlers to them, and connect them to scripts.

    `English` contains the English version of any localized information for `MainMenu.nib`. By default, the application does not contain localized information for any other languages, so clicking or double-clicking `English` has the same affect as clicking or double-clicking `MainMenu.nib`.

    Note: Localization is too complex a topic to describe here. For a description of how to localize resources, see the "Bundles" chapter of *Mac OS X Technology Overview*.

  - `InfoPlist.strings` contains strings representing information that is displayed to the user, such as the application name, version, and copyright information. For a description of how this information is used, see "Customize Version and Copyright Information" (page 214).

    `English` contains the English version of any localized string information. By default, the application contains only an English version, so clicking or double-clicking `English` has the same affect as clicking or double-clicking `InfoPlist.strings`.

- The Other Sources group contains the file `main.m`. This file contains the minimum Cocoa code required by the application. That code is shown in Listing 2-1 (page 59). You don't typically need to make any changes to this code file.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

- The Frameworks group contains two subgroups, Linked Frameworks and Other Frameworks. The Linked Frameworks group contains two frameworks:
  - `Cocoa.framework` The Cocoa framework includes both the AppKit and Foundation frameworks, which together provide the code and resources for Cocoa applications, including the user interface classes you use in AppleScript Studio. All AppleScript Studio applications link against this framework.
  - `AppleScriptKit.framework` This framework provides the scripting terminology and other extensions to Cocoa that allow AppleScript Studio applications to make use of Cocoa user interface classes. All AppleScript Studio applications link against this framework.

    For more information, see "AppleScriptKit Framework Overview" (page 60).

  The Other Frameworks group also contains two frameworks:
  - `Foundation.framework` As you've seen, the Cocoa framework includes the Foundation framework. Foundation is included in the Other Frameworks group as a convenience—by opening the framework, you can quickly access header and documentation files associated with it.
  - `AppKit.framework` As with the Foundation framework, the AppKit framework is included for convenience.

- The Products group contains the products produced by the targets in the project. The default project has just one target, which creates an application. In this example, `Simple Application.app` is the name of the application, where `.app` is the extension (which is not displayed in the Finder) for an application.

  The applications you build are saved in the build directory, which is usually a folder named "build" in the project folder. When you are ready to distribute your AppleScript Studio application, you copy it from this location.

Document-based Application Template

Figure 2-2 shows the Groups & Files list for a new document-based project (the Mail Search project, from a later tutorial in this document), with most groups expanded. Many of the items are common to all AppleScript Studio project templates, and are described in "AppleScript Application Template" (page 44). The following is a description of the default items that are unique to a document-based project:

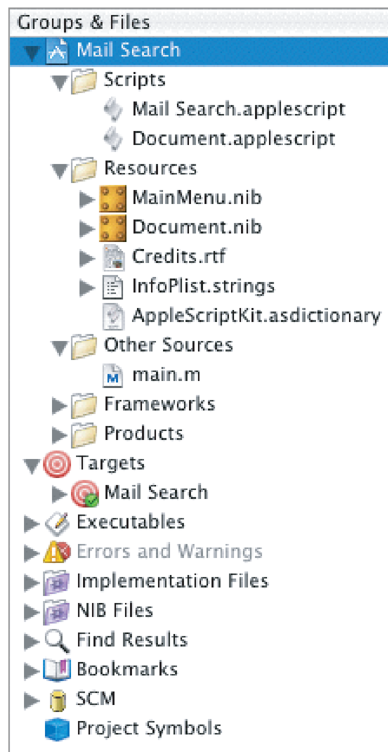
- `Document.applescript` is the default document script file, which is created as an empty file. If you have any document-related script statements, you should add them to this file.
- `Document.nib` is the nib file for creating document-associated windows. Nib files are described in "Interface Builder Features for AppleScript Studio" (page 53). You double-click the nib icon to open the file in Interface Builder.
- `Credits.rtf` is a rich text format (`.rtf`) file that supplies text for the default About window in a Cocoa application. You edit this file to supply your own About window information, as described in "Customize the About Window" (page 212).

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Note: In earlier versions of AppleScript Studio, the project contained a Classes group, which in turn contained two files, `Document.h` and `Document.m`. Those files contained the minimum code required to support documents in a document-based Cocoa application. Starting with AppleScript Studio version 1.2, they are no longer needed.

If you write Cocoa classes for your AppleScript Studio application, you can create a group for them.

Figure 2-2 Default contents of an AppleScript Document-based Application project



AppleScript Studio version 1.2 provides new terms for working with documents. These terms are documented in *AppleScript Studio Terminology Reference*, and you can also examine them in the Document suite in the AppleScript Studio scripting dictionary `AppleScriptKit.asdictionary`. For details on how to view this dictionary, see "Terminology Browser" (page 51).

AppleScript Droplet Template

AppleScript Studio provides the AppleScript Droplet template for creating its namesake—droplets. A **droplet** is a script application that launches when you drag a file or folder icon in the Finder and "drop" it on the droplet's icon. A droplet contains an `on open` handler, which receives a list of descriptors for the folders or files dropped on it. Droplets are a popular kind of application because they can conveniently handle everyday tasks, such as changing file extensions, printing, or performing more complicated operations, on each item in a list of files or folders.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

A project created with the AppleScript Droplet template contains the same items shown in Figure 2-1 (page 44) for the AppleScript Application template. The one difference is that the default script file, `Application.applescript`, is not empty in a droplet application. Instead, it contains the following handlers:

- `on idle`: Script statements you add to this handler perform idle time processing.
- `on open`: Script statements you add to this handler perform the droplet's main function.

  By default, the `open` handler ends with a `quit` statement. If you want the handler to stay open, remove the `quit` statement.

By adding script statements to these handlers, you can quickly create droplet applications.

The Targets Group

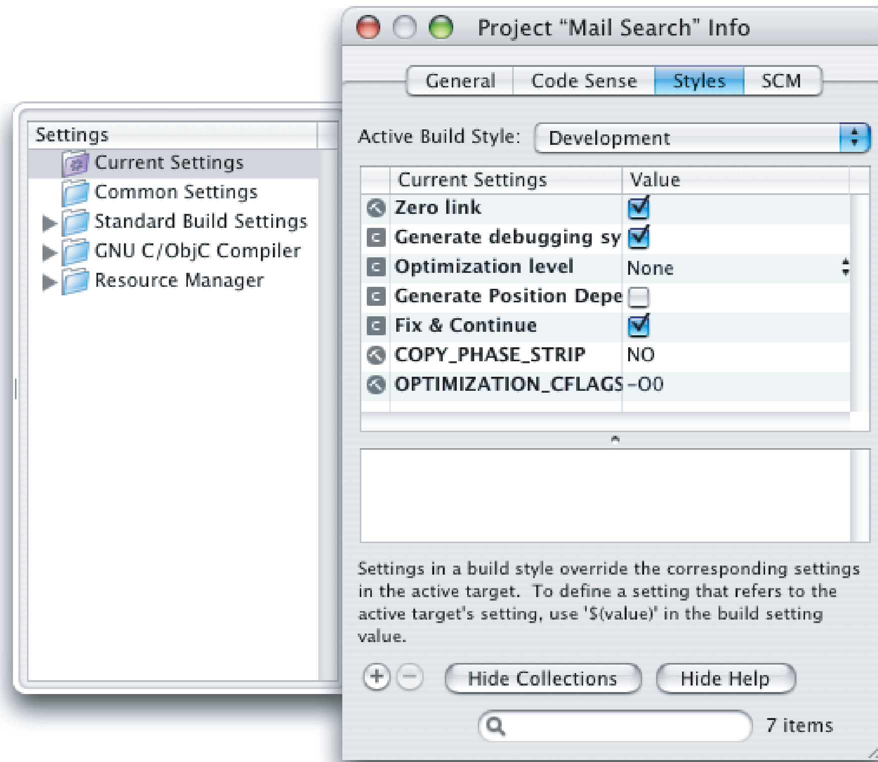
Xcode keeps track of project targets in the Targets group. A new AppleScript Studio project contains just one target, with the same name as the application (without the `.app` extension). For example, the default target for the project described in "Default Project Contents" (page 44) is "Simple Application". More complicated applications can have multiple targets.

Figure 2-3 shows the Targets inspector for the Mail Search application, developed in a tutorial later in this document. The figure shows the Styles pane, containing a pop-up menu to choose between Development and Deployment styles. By default, a new application uses the development build style. In a development build, debugging symbols are included and compiled script files in the application's bundle include the script text. In addition, ZeroLink is active. In a deployment build, symbols and script text are not included, and ZeroLink is turned off. As a result, a deployment build may have a much smaller disk footprint.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Note: ZeroLink is a feature you can read about in Xcode Help. You can create fairly complex AppleScript Studio applications without having to modify default settings.

Figure 2-3 The Mail Search Target inspector



Source Code Editor

Xcode provides a robust source code editor with many features, including:

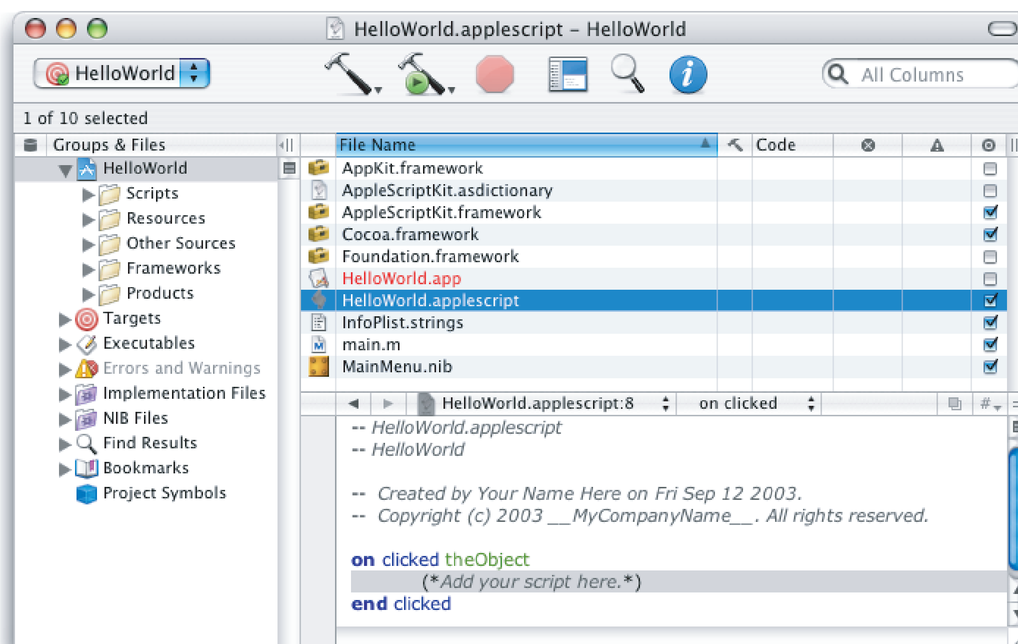
- options for switching between recently edited files, jumping to named handlers, and performing split-pane editing
- syntax checking for AppleScript, Java, C, and C++, as well as syntax coloring, which uses fonts, styles, and colors to distinguish, for example, between key words, comments, and so on
- automatic indenting and adjustable tab width
- matching of parentheses, braces, and brackets

When AppleScript Studio is installed, Xcode recognizes files that end in `.applescript` as script files. For example, the name of the default script file created for a new AppleScript Studio project is `Application.applescript`. When you select a script file in the Groups & Files list and click the Show Editor button, Xcode displays the file's contents in the main project window, as shown in Figure 2-4. (You can also double-click the file to open it in a separate editor window.) This window includes:

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

- a pair of arrows (the Go Back and Go Forward arrows) for switching between files displayed in the editor window
- a script icon, indicating the file is a script file (if you position the cursor over the script icon, it displays the full path to the script file)
- the name of the file (the name is in a pop-up menu; if you have displayed more than one file in the window, you can choose between them)
- a pop-up menu for navigating to any handler in the script (the current, and in this case only, handler is the `on clicked` handler)
- a "Go to Counterpart" button for switching between header and source files (not applicable for script files)
- a button for splitting the window into more than one pane (clicking again rejoins split panes

Figure 2-4 Editing a Hello World script in Xcode



You build an AppleScript Studio application with any of Xcode's mechanisms for starting a build, which include menu commands (such as Build in the Build menu), keystroke shortcuts (such as Command-B), and clickable buttons (any of the buttons that include a hammer). Building creates the application or other product specified by the current target.

When you build an AppleScript Studio application, all modified script files that belong to the current target are compiled automatically. You can compile an individual script in an editor window by typing Command-K or pressing the Enter key.

Building the application also compiles the Cocoa code in the application's `main.m` file (shown in Figure 1-10 (page 29)) and in any other source files in the current target, and links with the application's frameworks. For more information on building applications, see any of the tutorials in this document, or refer to the Documentation Window in the Xcode Help menu.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

In Figure 2-4 (page 50), AppleScript keywords are shown in blue (`on`, `end`) and application keywords in red (`clicked`, `display dialog`). For information on script formatting, see "How Xcode Formats Scripts" (page 69).

You can read more about the standard features available with Xcode's source code editor in the Documentation Window, found in the Xcode Help menu.

Debugging Features

Note: AppleScript Studio's support for line-by-line debugging in Xcode is still in development. This document will be updated to describe this support as it becomes available.

The following sections contain some information that can help you debug your application:

- "Programming Tips" (page 80) provides pointers for creating Studio applications. The section "Using the Log Command to Track Your Scripts" (page 80) can be useful in debugging your scripts.
- "Troubleshooting" (page 82) provides a list of common problems and tips for solving them.

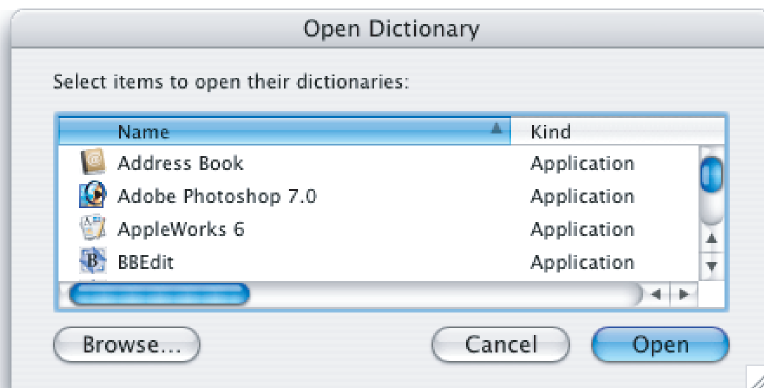
You may be also able to debug your application with a third-party debugger.

Terminology Browser

Xcode provides the Open Dictionary command in the File menu to examine the scripting dictionary of any scriptable Carbon or Cocoa application. The dictionary defines the application's scripting terminology—the English-like words and phrases you can use in a script to communicate with and control that application.

The Open Dictionary command brings up a dialog similar to the one in Figure 2-5, where you can select from among the currently available scriptable applications.

Figure 2-5 The Open Dictionary dialog in Xcode



2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Within an AppleScript Studio project, you can display the AppleScript Studio scripting dictionary by double-clicking the icon for `AppleScriptKit.asdictionary` in the Files list in Xcode's Groups & Files list. The result is a browser window similar to the one shown in Figure 2-6. If you merely click the icon, or if you choose an AppleScript Studio application with the Open Dictionary command, the browser is displayed as a pane within the project window. In either case, you have access to the same information and navigation options.

Note: The color of text you see when you display `AppleScriptKit.asdictionary` depends on the color settings at the time the file was created, and may vary from what is shown in Figure 2-6.

You can also open dictionaries in the Script Editor application. Both applications provide the same information.

Figure 2-6 The AppleScript Studio scripting dictionary in a browser window

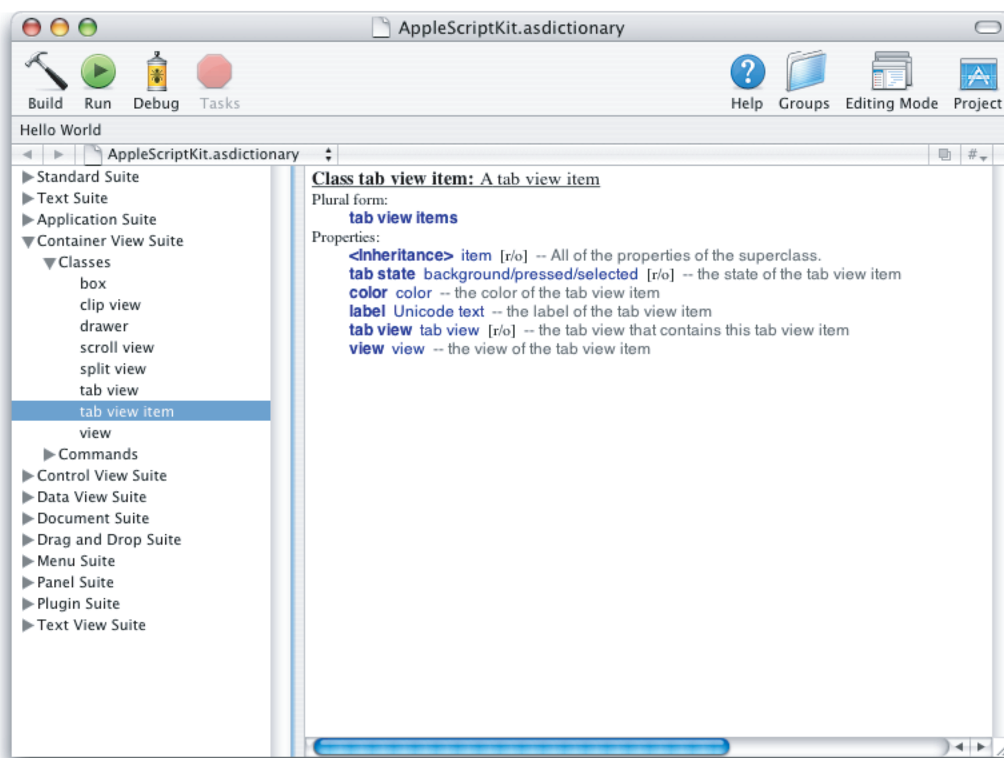


Figure 2-6 shows the classes (which can include elements and properties) and events (which provide syntax descriptions) of the Container View Suite, with the terminology for the tab view item class currently visible.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Note: The Event container in the scripting dictionary can contain both commands and events. A **command** is a word or phrase a script can send to an object to request an action. For example, a script can send a `stop` command to a progress indicator.

An **event** is an action an object can respond to. For example, a button click is an event that may result in execution of a `clicked` handler for the button that was clicked.

More simply, scripts can send commands to objects, while events, often the result of user actions, generate calls to event handlers in scripts.

Figure 2-6 also shows a number of suites of related terminology. The Standard and Text suites are part of Cocoa and are available to all Cocoa applications that turn on scripting support. The remaining suites are provided by AppleScript Studio, which groups its scripting terminology in these suites. See *AppleScript Studio Terminology Reference* for detailed information about the available terminology, and also the section "AppleScript Studio Terminology" (page 73) in this document.

Note: Outline items in the dictionary viewer shown in Figure 2-6 do not remember their expanded state, so if you shift to display another file in the window, when you shift back to the dictionary viewer all of the items will be collapsed. To avoid this, open the dictionary viewer in its own window. You can do so by double-clicking the file `AppleScriptKit.asdictionary` in the Files list in an AppleScript Studio project's Groups & Files list.

Interface Builder Features for AppleScript Studio

Interface Builder is Apple's graphical interface builder for Mac OS X. Interface Builder lets you lay out interface objects (including windows, controls, menus, and so on), resize them, set their attributes, and make connections to other objects. The resulting information is stored in user interface resources, called **nibs**, which in turn are stored in **nib files** that become part of your application. (A nib file is an Interface Builder file—the "ib" in "nib" stands for Interface Builder.) When the application is opened, it creates an interface containing the windows, buttons, and other user interface objects specified in its nib files.

The Interface Builder application is located in `/Developer/Applications`. Interface Builder has extensive online help and release notes. There are also tutorials available that describe how to build interfaces for Carbon and Cocoa applications.

Interface Builder's support for AppleScript Studio includes the following features:

- Graphical tools for creating sophisticated interfaces (not limited to AppleScript Studio applications). Some of these tools are reviewed in "Interface Creation" (page 54). For additional documentation, see the tutorials throughout this document, as well as the tutorials and online help for Interface Builder.
- Access to Cocoa's rich set of user interface objects (also not limited to AppleScript Studio applications). For more information, see "Cocoa User Interface Objects" (page 59).
- An AppleScript palette, which provides custom AppleScript Studio objects. For more information, see "Interface Creation" (page 54).
- An AppleScript pane in the Info window. You can use this pane to connect user interface objects to scripts—when an event occurs, such as a user clicking a button, the application calls a specified event handler in a script. For more information, see "Interface Connections" (page 56).

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

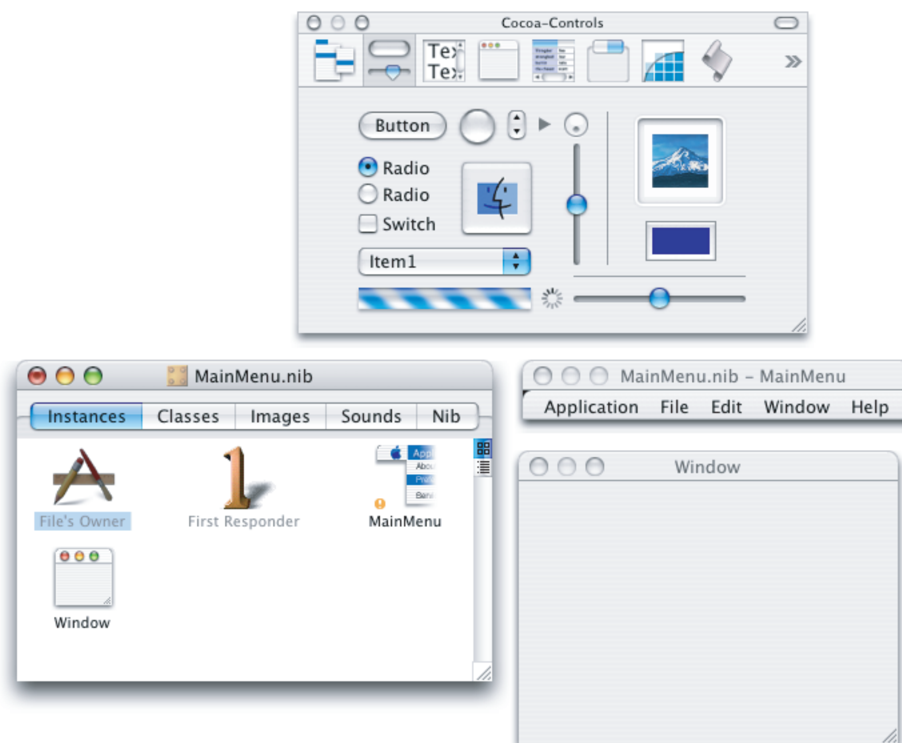
The following sections describe these features in more detail.

Interface Creation

To create an interface with Interface Builder, you create and edit a nib file containing descriptions of the interface elements in your application. A nib file can describe all or part of a user interface. Many applications use multiple nib files—for more information, see "Deciding How Many Nib Files to Use" (page 63).

When you create an AppleScript Studio application with Xcode, it automatically contains a default nib file named `MainMenu.nib`. The icon for this nib file is shown in Figure 2-1 (page 44). Double-clicking the icon for a nib file opens Interface Builder. When you open the default `MainMenu.nib` file for an AppleScript Studio application, Interface Builder opens four windows, such as those shown in Figure 2-7.

Figure 2-7 Interface Builder windows after opening the `MainMenu.nib` file



The `MainMenu.nib` window displays the Instances pane, showing four instances, two of which (MainMenu and Window) are shown in their own windows. The four instances are:

- File's Owner: Every nib file has one owner, an object outside the nib file that relays messages between objects that are created from the nib. In this, the main nib file, File's Owner always represents `NSApp`, a global constant that references the NSApplication object for the application. The application object serves as the master controller for the application. For more information

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

on the application object, see "Cocoa Application Framework" (page 59). For more on File's Owner, see "Connect the Application Object" (page 169). For more detail, see the Cocoa documentation described in "See Also" (page 15).

- First Responder: This instance identifies the object that is the first target in the application for keystrokes. You won't typically need to modify this instance in AppleScript Studio applications.

> **Note:** There is a window property `first responder` you use to set the current focus; for example `set first responder of window 1 to text field 1 of window 1` sets the focus to the specified text field so that it will receive keystrokes. As of AppleScript Studio version 1.2, you can only set the `first responder` property; getting it will not return a useful value.

- MainMenu: This instance defines the menus for the application. It is displayed in its own window in Figure 2-7. For information on working with menus, see "Customize Menus" (page 209).
- Window: This instance defines the document window for the application, and is also displayed in its own window in Figure 2-7. For information on working with windows, see "Create the Message Window" (page 137) and "Create the Search Window" (page 152).

For information on the other panes visible in the MainMenu.nib window, see Interface Builder Help.

The window containing a number of buttons and fields is the Palette window. You click the buttons in the Palette window's toolbar to display other palettes, such as the AppleScript palette (whose button is on the left of the toolbar).

> **Note:** Interface Builder allows you to customize the toolbar, so it may not appear exactly as shown in Figure 2-7. To customize, click in the toolbar while holding down the Control key to see a menu of options.

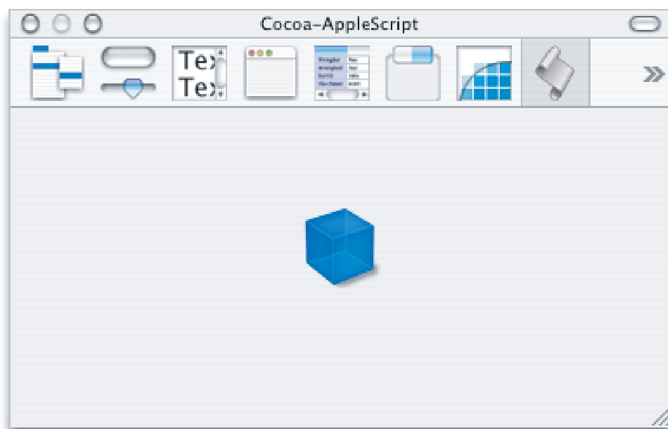
To add Cocoa user interface objects to your application's main window, you simply drag them from the Cocoa-Controls palette (or from another palette) and position them in the Window window.

To create a new Window, you drag a window out of the Cocoa-Windows palette and release it. The window will appear both as an instance in the nib window and as a window, into which you can drag user interface items.

One feature of the Palette window that is unique to AppleScript Studio is the AppleScript palette, shown in Figure 2-8. This palette provides access to special-purpose objects the AppleScriptKit framework supplies for use in AppleScript Studio applications. For example, the icon in Figure 2-8 represents a **data source object** that supplies data to a table view or other view with rows and columns. (The data source icon shown is subject to change.) The section "Connect the Search Window" (page 173) in the Mail Search tutorial defines the data source and describes how to work with data source objects.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Figure 2-8 The AppleScript palette in Interface Builder's Palette window



Interface Builder provides several mechanisms for modifying interface items:

- You can drag to move or resize items; as you drag, Interface Builder provides feedback to help align items according to the Aqua interface guidelines.
- For many items, such as buttons, you can double-click the item to select its title, then type a new title.
- You can use commands from the Format, Layout, and Tools menus to modify and position user interface items.
- You can select a user interface item, then open the Info window (by typing Command-Shift-I or choosing Show Info from the Tools menu). The Info window has a number of panes, selected through a pop-up menu, which allow you to set attributes, adjust size, and perform other operations. The AppleScript pane is shown in Figure 2-9 and described in "Interface Connections" (page 56).

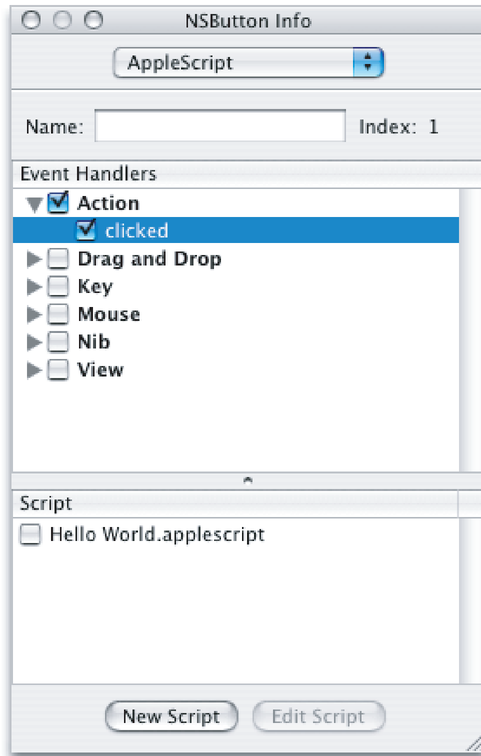
Interface Connections

Interface Builder provides the AppleScript pane in the Info window to connect objects in an application's user interface to application scripts. You create connections so that when a user performs an action, such as typing text, clicking a button, or choosing a menu command, the application calls the appropriate handler to deal with that action.

To connect a handler, you select a user interface item in an Interface Builder window, then open the Info window by typing Command-Shift-I or choosing Show Info from the Tools menu. Use the pop-up menu at the top of the window to choose the AppleScript pane. Figure 2-9 shows the Info window for a button in the Hello World application, described in "Creating a Hello World Application" (page 27).

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Figure 2-9 The Info window for a button



The AppleScript pane provides the ability to

- select any available event handlers for an object

  The Info window shows all the event handlers available to objects of the current class (in this case, the button object is an instance of NSButton). That may include handlers inherited from superclasses.

  Event handlers are grouped by function. You click the checkbox for any group to connect all handlers in that group, or click one or more checkboxes within a group to connect individual handlers.

- assign selected event handlers to an existing script or to a new script you create

  Any existing scripts in your application appear in the Script list. You click the New Script button to create a new script, which is added to the list of scripts.

  To assign a handler to a script, check the handler, select the checkbox for the script, then click the Edit Script button. This automatically switches to an editor window in Xcode for the selected script. Xcode inserts an empty handler for the event, such as the following `clicked` handler:

  ```
  on clicked theObject
      (*Add your script here.*)
  end clicked
  ```

  You can then add any desired script statements to the handler.

- identify objects by an automatically assigned index, or by a name you supply

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

The button object in Figure 2-9 has index 1, meaning it is the first button in the window. You can also type a name in the Name field. Then, from within your scripts, you can refer to the object by either name or index. You can also manipulate objects in scripts by ID, but AppleScript Studio and Interface Builder do not supply a mechanism for obtaining the ID while creating the interface.

Note: Identifying an object in a script by its unique ID is generally safer than using an index number, which can change during script execution.

You can use the AppleScript pane in the Info window to examine the event handler definitions for all available handlers for a particular object. Just select all the checkboxes, choose a script, and click the Edit Script button. Xcode inserts an empty handler for every selected event.

As mentioned in "Interface Creation" (page 54), the Info window provides panes for performing many operations, such as setting the enabled state and other attributes of an object. The tutorial chapters in this document describe how to perform operations with the Info window, and you can also read about those operations in Interface Builder Help. It is possible to use the Info window to hook up objects to Cocoa class methods. See the Cocoa documentation described in "See Also" (page 15) for more information.

Cocoa Framework Overview

While a previous section described AppleScript as the driving force behind AppleScript Studio, the Cocoa application framework provides the key technology for creating AppleScript Studio applications and supplying their user interfaces. This section provides a brief overview of features of the Cocoa framework.

The **Cocoa framework** is an object-oriented application framework. It pulls together two other frameworks, the AppKit and Foundation frameworks. Together, the classes and resources in these frameworks provide the basic building blocks for sophisticated Mac OS X applications.

The **Foundation framework** defines a layer of useful primitive object classes, including support for Unicode strings, allocation and deallocation of objects, arrays and collections, dates, ports, and more. The **AppKit framework** provides classes for a graphical, event-driven user interface, including windows, buttons, text fields, and more.

These frameworks are located in `/System/Library/Frameworks`, along with the other frameworks available with Mac OS X.

Cocoa Scripting Support

Cocoa applications can take advantage of automated scripting support supplied by the Foundation and AppKit frameworks. With the help of a scripting definition supplied by the application itself, this support converts incoming Apple events into script command objects that access application objects automatically to perform the specified operation. This mechanism makes it very easy to provide basic scripting support, such as responding to Apple events by manipulating scriptable objects and data in the application.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

To turn on Cocoa's built-in scripting support, an application must link with the AppKit and Foundation frameworks, and must also add a key to its `Info.plist` file:

```
NSAppleScriptEnabled = YES
```

When you choose an AppleScript Studio project template in Xcode, this key is automatically added to your application's `Info.plist` file.

AppleScript Studio uses an additional framework, the AppleScriptKit framework, to enhance the built-in scriptability available to Cocoa applications. That framework is described in "AppleScriptKit Framework Overview" (page 60). For information on how to use this framework with your Cocoa application, see "Adding AppleScript Studio Support to Your Cocoa Application" (page 86).

Cocoa User Interface Objects

Cocoa provides a wide variety of user interface objects for use in your application. These objects range from the basic (buttons, checkboxes, text fields) to the sophisticated (windows, panes, tabs, browsers). You add these items to your application's user interface with Interface Builder, as described in "Interface Creation" (page 54). That section describes how to view the available objects in Interface Builder's Palette window, shown in Figure 2-7 (page 54).

Another way to examine the user interface objects available in AppleScript Studio is to build the various applications distributed with AppleScript Studio. These applications are described in "AppleScript Studio Sample Applications" (page 35).

The user interface objects you use in AppleScript Studio are instances of Cocoa classes defined in the AppKit framework. These classes are part of an application framework that provides many sophisticated features for creating object-oriented applications. For more information, see the Cocoa documentation described in "See Also" (page 15).

Scripters can access these user interface classes through the terminology provided by AppleScript Studio. "AppleScript Studio Terminology" (page 73) describes AppleScript Studio's script suites, which specify the available objects and events you can use in AppleScript Studio scripts.

Cocoa Application Framework

Because Cocoa is a full-featured application framework, just building the default application from a Cocoa project template results in an application that is ready to do real work, including displaying its interface and responding to user actions.

AppleScript Studio applications take advantage of the Cocoa framework, which works "behind the curtain" to display the interface, respond to user actions, and more. But there is very little visible Cocoa code required for an AppleScript Studio application (see Listing 2-1). As a result, scripters gain the ability to create complex interfaces, work in a powerful development environment, and control applications using AppleScript script statements.

Listing 2-1 Full Objective-C code for a simple AppleScript Studio application

```
extern int NSApplicationMain(int argc, const char *argv[]);

int main(int argc, const char *argv[])
{
```


2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

```
    return NSApplicationMain(argc, argv);
}
```

In Listing 2-1, the main function calls `NSApplicationMain` to create an application object, load the application's main nib file (thus creating the interface), and call the application object's `run` method. The application object's main task is to receive events and distribute them to the objects in the application that should respond to them. For example, all keyboard and mouse events go directly to the window object associated with the event. In an AppleScript Studio application, these events can then be dispatched to script event handlers (described in the next section) associated with user interface objects.

Although you can create applications that perform virtually all of their operations by executing AppleScript scripts, you are free to include additional Cocoa code in applications. And if you want to know more about the Cocoa code working behind the scenes, see the Cocoa documentation described in "See Also" (page 15).

AppleScriptKit Framework Overview

Any Cocoa application can take advantage of automated scripting support supplied by the Foundation and AppKit frameworks. AppleScript Studio uses an additional framework, the **AppleScriptKit framework**, to supply advanced Cocoa scripting support that allows AppleScript Studio applications to work with Cocoa user interface objects in scripts. All AppleScript Studio applications automatically link with this framework.Cocoa applications can also link with it, as described in "Adding AppleScript Studio Support to Your Cocoa Application" (page 86).

When you install AppleScript Studio, the AppleScriptKit framework is installed in `/System/Library/Frameworks`. The framework is also installed automatically with Mac OS X version 10.1.2 and later, so even users who haven't installed AppleScript Studio can use your Studio applications if they have installed the corresponding version of Mac OS X. See "AppleScript Studio System Requirements and Version Information" (page 221) for details on how to determine what version (if any) of AppleScript Studio is installed.

The primary importance of `AppleScriptKit.framework` to scripters is that it supplies the scripting terminology you use to access user interface objects in AppleScript Studio scripts. You can read about this terminology in "AppleScript Studio Terminology" (page 73) and find instructions on how to display it in "Terminology Browser" (page 51). For very detailed terminology reference, see *AppleScript Studio Terminology Reference*.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Programming With AppleScript Studio

This chapter describes a number of features and issues that will help you get the most from AppleScript Studio. It contains the following sections:

- "Additional Information on AppleScript Studio" (page 61)
- "AppleScript Studio Terminology" (page 73)
- "Programming Tips" (page 80)
- "Troubleshooting" (page 82)

For related information, see "Strengths and Limitations" (page 20).

Additional Information on AppleScript Studio

The following sections describe additional features and issues you'll want to know more about as you work with AppleScript Studio.

- "Organizing an AppleScript Studio Project" (page 62)
- "Naming Conventions for Methods and Handlers" (page 64)
- "Accessing Code From AppleScript Studio Scripts" (page 64)
- "Persistent Script Properties" (page 67)
- "Accessing Script Globals" (page 68)
- "Overridden Scripting Additions" (page 68)
- "How Xcode Formats Scripts" (page 69)
- "Switching Between AppleScript Studio and Script Editor" (page 70)
- "Scripting AppleScript Studio Applications" (page 71)
- "Using Script Editor to Test AppleScript Studio Terminology" (page 72)

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Organizing an AppleScript Studio Project

Two questions you may frequently face in organizing an AppleScript Studio project are whether to use one or many script files and whether to use one or many nib files. In each case, the answer depends on the scope and goals of the project.

Deciding How Many Script Files to Use

When you create a new AppleScript Studio project in Xcode with the AppleScript Application or Droplet templates, it contains one script file, `Application.applescript`. If you use the Document-based template, you get an additional script file, `Document.applescript`. As the names suggest, these script files are intended for handlers related to the application and its documents, respectively. However, you are free to delete these script files, to rename them, or to add additional script files.

Given this freedom of choice, how should you organize the handlers and other script statements you write for an AppleScript Studio application? As you might expect, the answer depends on the scope of the project and the complexity of the user interface.

There are several advantages to putting all of your script statements in one file:

- Because they are in one file, handlers and script objects have access to other handlers, script objects, and global properties in the file. When this access is important, use of one file makes sense. Script objects are described in "Additional Handlers and Scripts in Mail Search" (page 129).
- There is less overhead with a single script file. For a small application or one with a simple user interface, creating multiple script files may slow the pace of development.

A significant disadvantage of using a single script file is that if many similar objects in the interface (such as buttons) share a handler (such as the `clicked` handler), you may need to do lengthy testing to determine which object the handler was called for. An example is shown in Listing 3-1. Thus using a single script file can have drawbacks in the case of a complex interface with many similar objects, such as a preferences panel. It can also lead to greater complexity in testing and debugging.

Listing 3-1 Detecting which button was clicked

```
on clicked theObject
    if the name of the object is "Dial button" then
        --do something
    else if name of the object is "Hang Up button" then
        --do something else
    else if name of the object is "Panic button" then
        --do a third thing
    else if ...
```

There are also advantages to using multiple script files:

- Modularity is a widely-accepted principle in software development. Grouping like things together can make the application both easier to understand and easier to test and debug.
- Having one script per significant object allows you to avoid the code complexity shown in Listing 3-1. Within a handler, you know which object triggered the call. In fact, you should put a comment to that affect in the handler itself.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

The downside to using multiple script files is a proliferation of small files in the project. That makes multiple script files most appropriate for projects with a significant, but manageable, number of similar user interface objects.

Finally, you may want to provide one script file per window in your application. This approach may make sense if you are using a similar approach for nib files (as described in the next section).

Deciding How Many Nib Files to Use

To create an interface with Interface Builder, you create and edit a nib resource file that contains descriptions of the interface elements in your application. A nib file can describe all or part of a user interface. Many applications use two or more nib files, with one of them designated as the main nib file. The main nib file contains the main menu and any windows and panels you want to appear when your application starts up.

In addition to the main nib file, you can have one or more nib files that you load whenever you need them. Loading a nib file unarchives (or creates instances of) whatever user-interface objects are described in the nib. For example, if your application creates its own document type, you might have a separate nib file for a document window. Each time a user opens a new document, you would create a document window by loading the auxiliary nib file.

It is certainly possible to put a large number of user interface definitions into a single nib file. However, as you add object instances (and possibly classes, images, and sounds as well) to a nib file, the task of working with the nib in Interface Builder becomes more complicated. For example, you can use Interface Builder to examine the objects in a nib file in an outline view, as described in "Examining an Object Hierarchy in the Nib View" (page 182). This mode of display can be very useful for examining an object hierarchy and viewing the connections between objects. However, as you add objects to the nib file, the clarity of the hierarchy and relationships diminishes.

As a result, using a single nib file probably only makes sense for relatively small AppleScript Studio applications with simple user interfaces, and for applications that are not built with the Document-based project template.

A rule of thumb for creating nib files is to use one nib for each separate kind of window in the application. For example, the Mail Search application, described in detail in the tutorial beginning in "Chapter 7, Mail Search Tutorial: Design the Application," (page 119) uses four nib files: one for the application and its menus, one for the search window, one for the message window, and one for a status dialog. By using a single nib for a window definition, you can easily create instances of that window object by loading the nib with the `load nib` command.

> **Note:** When you load a nib file, Cocoa instantiates all the top level objects archived in the nib. So if you have more than one window defined in a nib file, loading the nib file will instantiate each window. That may be appropriate if your application has several windows that you always want to instantiate on launch, and not again thereafter. And you can use a window object's `visible` property to control when the window is visible.

One final advantage of using multiple nib files is that doing so can help simplify the task of finding and correcting interface-related bugs—and in AppleScript Studio, the interface is likely to be a major factor in most applications.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Naming Conventions for Methods and Handlers

The Cocoa application framework follows a naming convention that helps explain when certain methods are called. This convention, which is reflected in the terminology for AppleScript Studio's event handlers, inserts `should`, `will`, and `did` in method names. Table 3-1 describes the meaning of these terms. Note that to indicate a completed operation, AppleScript Studio uses the past tense, rather than the term `did`.

Table 3-1 Naming conventions in Cocoa and AppleScript Studio

Cocoa phrase	Explanation	AppleScript Studio examples
`should`	Asks whether an operation should take place. You can cancel the operation by returning `false`.	`should open` `should close`

`will`	An operation is about to take place. You can prepare for it, but not prevent it.	`will resize`
		`will hide`
		`will quit`

`did`	An operation has completed. You can perform actions in response to it. AppleScript Studio uses past tense, rather than the term `did`.	`activated`
		`launched`
		`miniaturized`
		`zoomed`

So, for example, you can add a `should close` handler to a window object. When the handler is called, it can determine whether the user has performed some essential task—if not, it can return `false` and refuse to allow the window to close. A `will close` handler cannot cancel the close operation, but it can perform any necessary tasks to prepare for closing. Finally, a `closed` handler can perform any tasks required after closing.

See *AppleScript Studio Terminology Reference* for detailed descriptions of the event handlers and other terminology that is available in AppleScript Studio.

Accessing Code From AppleScript Studio Scripts

AppleScript Studio provides the `call method` command for calling methods of Objective-C objects in an AppleScript Studio application.

The `call method` command provides the ability to:

- target user interface objects
- target the application object or its delegate
- specify as many parameters as needed
- receive a return value; the return value can be another object, from which you can extract more information

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Because you can access other languages from Objective-C, the `call method` command also allows you to:

- access code written in C, C++, Objective-C++, and Java (both directly and through the Java bridge—a Mac OS X mechanism for communicating between Objective-C and Java)
- access legacy code written in one of these languages
- access Mac OS X frameworks, such as the Core Foundation and Carbon frameworks

The Multi-Language application, distributed starting with AppleScript Studio 1.1, demonstrates how to call other languages from an AppleScript Studio application.

Figure 3-1 shows the syntax for the `call method` command.

Figure 3-1 Syntax for the call method command

```
call method: calls the given method of the object with the specified parameters
    call method reference -- the object for the command
        [of anything] -- the object to send the method to (exclusive from the "of class" parameter)
        [with parameter anything] -- a parameter to be passed to the method (exclusive of "with parameters")
        [of class Unicode text] -- the class to send the method to (exclusive from the "of object" parameter)
        [of object anything] -- deprecated in favor of the "of" parameter
        [with parameters list] -- a list of parameters to be passed to the method (exclusive of "with parameter")
```

`call method`
: Invokes the command, with `reference` specifying the method to call.

`with parameter`
: This optional parameter allows you to pass a value to a method that takes a single parameter. You can use the parameter to pass an object or a simple value such as an integer. You can also pass a single list, which can contain multiple items, but only if the called method expects a single parameter that encompasses multiple values, such as an array or dictionary.

`of class`
: This optional parameter allows you to specify the Objective-C class whose method is called.

`of object`
: This optional parameter allows you to specify the object whose method is called.

: You never use both `of object` and `of class`. If you don't specify either, the call goes to a method of the application's delegate object or, if the delegate doesn't support it, to the application object itself.

: The application object is described in "Cocoa Framework Overview" (page 58). For information on class methods, delegates, and other Cocoa topics, see the documentation described in "See Also" (page 15).

`with parameters`
: This optional parameter is intended for use with methods that have more than one parameter, though you can also use it for a method with a single parameter. You specify a list with one item for each parameter of the specified method. An item within the list of parameters can be a list, if the called method expects a single parameter that encompasses multiple values in that position.

: You never use both `with parameter` and `with parameters`. If you don't use either, it is assumed the method has no parameters.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Note: If a parameter name is the same as an AppleScript keyword, you can enclose it between vertical bar characters (for example, `|Set|`) to prevent AppleScript from evaluating it and potentially changing the case during compilation.

It is useful to note that Objective-C associates a colon with each parameter of a method. For example, the following is a method declaration from Cocoa's NSDocument class:

```
- (BOOL) readFromFile: (NSString *) fileName ofType: (NSString *) docType
```

This method has two parameters, so to call it with call method, you use the `with parameters` option, as shown in listing Listing 3-2:

Listing 3-2 Calling a document method with two parameters

```
call method "readFromFile:ofType:" of object (document 1 of window 1)
    with parameters {myFilenameString, myDocTypeString}
```

In Listing 3-2, the list consists of the two string variables (whose values are set prior to the call) enclosed in curly brackets: `{myFilenameString, myDocTypeString}`.

For a method with one parameter (and one colon), you use `with parameter`. The example in Listing 3-3 calls the `performClick:` method of a button object, passing as a parameter another button object.

Listing 3-3 Calling a method of a button

```
call method "performClick:" of object (button 1 of window 1)
    with parameter (button 2 of window 2)
```

The single parameter in Listing 3-3 is enclosed in parentheses because it is a multi-term reference: `(button 2 of window 2)`. You could optionally use `with parameters` and pass a one-item list: `{button 2 of window 2}`.

Listing 3-4 shows an example that calls a class method of NSNumber to get back a number object initialized with an integer value. It passes a single value (the number 10) for its one parameter. In this case, the parameter is unambiguous, and does not require parentheses.

Listing 3-4 Calling a class method

```
set theResult to call method "numberWithInt:" of class "NSNumber"
    with parameter 10
```

Listing 3-5 shows a hypothetical example that has no parameters. Because it doesn't specify a class or object, call method will attempt to execute the `countMyCustomers` method of the application object, returning the customer count. It's your obligation to make sure the method you call actually exists!

Listing 3-5 Calling a method of the application

```
set customerCount to call method "countMyCustomers"
```

The `call method` command can accept and return the types NSRect, NSPoint, NSSize, and NSRange, in addition to primitive types such as `int`, `double`, `char *`, and so on. For example, to call the NSView method `- (void) setFrame: (NSRect) frameRect`, you use a script statement similar to the following:

```
call method "setFrame:" of object (view 1 of window 1)
```


2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

```
        with parameter {20, 20, 120, 120}
```

In this case, the single parameter is a 4-item list that is passed to `setFrame` as a type NSRect.

Table 3-2 lists Cocoa types you typically use with the `call method` command and their AppleScript equivalents.

Table 3-2 Cocoa types and their AppleScript equivalents

Cocoa type	AppleScript equivalent
NSArray	list
NSDate	date
NSDictionary	record
NSPoint	list of two numbers: {x, y}
NSRange	list of two numbers: {begin offset, end offset}
NSRect	list of four numbers: {left, bottom, right, top}
NSSize	list of two numbers: {width, height}
NSString	string

To see script statements that invoke the `call method` command in a working project, see the Archive Maker or Drawer sample projects, described in "AppleScript Studio Sample Applications" (page 35).

Persistent Script Properties

In AppleScript Studio, script properties are not saved back into the application as they currently are in AppleScript script applications created with Script Editor or other script editing applications. Therefore the values of script properties do not persist between launches of an AppleScript Studio application.

If you want persistent storage of values, you can write them to a preferences file before your application quits and read them back when it is launched. Starting in version 1.1, AppleScript Studio also provides a mechanism for conveniently saving and restoring values from scripts using the user defaults system available in Mac OS X. This mechanism is described in *AppleScript Studio Terminology Reference*—see the `user-defaults` class and the `default entry` class.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Accessing Script Globals

In AppleScript Studio, global variables declared in one script are not accessible from another script without doing an explicit `load script` command. (See the Unit Converter sample application for an example of how to load a script.) Even when you load a script, you're getting a snapshot of the current values of the variables, not access to one variable that can be referenced (and updated) by each script.

If you need to keep one set of values that can be accessed and updated from multiple scripts, use one of the mechanisms described in "Persistent Script Properties" (page 67).

Overridden Scripting Additions

AppleScript Studio overrides the `display dialog` command to provide its own version, which can be displayed as a sheet (a dialog attached to a window).

Note: The `display dialog` command is part of AppleScript's Standard Additions scripting addition, located in `/System/Library/ScriptingAdditions`.

When specifying a `display dialog` command, you use a term similar to the following to display the dialog as a sheet with the specified window:

```
display dialog ... [other terms] ... attached to window "Window Name"
```

Important: When you work with a dialog displayed as a sheet, you must supply a `dialog ended` handler to continue processing when the user dismisses the sheet. For details, see the Discussion session for `display dialog` command, in the "Panel Suite" section of *AppleScript Studio Terminology Reference*.

For a detailed example of how to use AppleScript Studio's version of `display dialog`, see the Display Dialog sample application (distributed with AppleScript Studio). The following call to the `display dialog` command is from that application. Many of the parameters are variables, set before making the call.

```
display dialog dialogText buttons
        {dialogButton1, dialogButton2, dialogButton3}
    default button dialogDefaultButton
    giving up after dialogGivingUpAfter
    with icon dialogIcon attached to window "main"
```

If you pass a string for the `with icon` parameter, the command will look for a `.tiff` resource with that name.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Important: The `with icon` parameter will accept a number, with 0, 1, and 2 corresponding to the stop, note, and caution icons, respectively. However, use of these icons is no longer recommended. In addition, the AppleScript constants `stop`, `note`, and `caution` do not work.

The `display dialog` command generates a "user canceled" error when the cancel button is pressed only if the dialog is not attached to a window. If the dialog is attached, cancel is treated like any other button, and you must check for it in your `dialog ended` handler. The Display Dialog sample application demonstrates how to display dialogs, both as a plain dialog or as a sheet attached to a window.

For more information on `display dialog`, see the "Panel Suite" section in *AppleScript Studio Terminology Reference*.

How Xcode Formats Scripts

When you check the syntax for a script, Xcode reformats it according to your formatting preferences. Xcode's editor window currently uses the same formatting (font, style, and color) you have specified in the Script Editor application (located in `/Applications/AppleScript`). To change settings in the version of Script Editor released with Mac OS X version 10.3, follow these steps:

1. Quit Xcode.
2. Open the Script Editor application (located in `/Applications/AppleScript`).
3. Choose Preferences… from the Application menu.
4. Click the Formatting button.
5. Choose the fonts and styles you prefer. (Or click the Use Defaults button to go back to the default values.)
6. Click the Apply button, then quit Script Editor.
7. Open Xcode. Your formatting changes are now in effect.

Because of differences in Script Editor and Xcode, the same font may be rendered differently in the two applications. For an example of script formatting, see Figure 2-4 (page 50). In that script, AppleScript keywords are shown in blue (`on`, `end`) and application keywords in red (`clicked`, `display dialog`).

Note: When you first install AppleScript Studio, editor windows in Xcode may not reflect settings from Script Editor's AppleScript Formatting menu until after you've made some change to those settings and restarted Xcode.

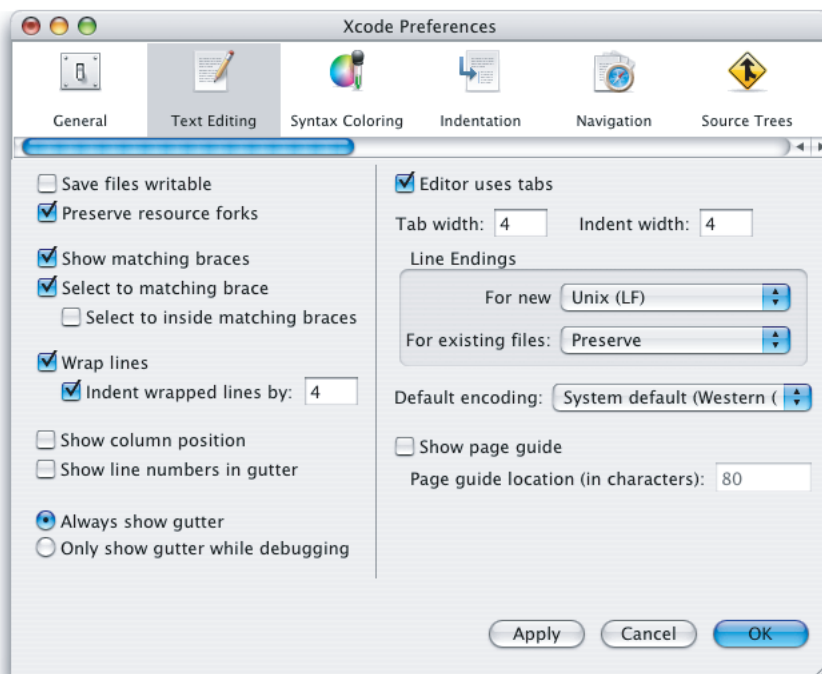
Most scripters are familiar with AppleScript's line-continuation character, "¬", used for dealing with long lines. When a line ends with that character, AppleScript treats the following line as part of the previous line. Long lines are common with some of the user interface terminology in AppleScript Studio. However, Xcode's ability to wrap lines provides an elegant way to deal with long lines without using continuation characters. You can also set tab widths to improve control script formatting.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

To adjust line wrap and tab settings, choose Preferences in Xcode's application menu, then click the button for the Text Editing pane, shown in Figure 3-2. To choose settings that work well for AppleScript Studio projects:

- select the checkbox for "Wrap lines"
- select the checkbox for "Indent wrapped lines by:" and enter a value of 4
- select the checkbox for "Editor uses tabs"
- enter a value of 4 for "Tab width:" and "Indent width:"

Figure 3-2 Setting Text Editing preferences in Xcode



Switching Between AppleScript Studio and Script Editor

If you copy text from an AppleScript Studio script file to a Script Editor window on a version of Mac OS X prior to version 10.3, you may see the script displayed with many "garbage" characters. The strange characters are probably there because Script Editor can not deal effectively with UNIX-style line endings, while that is the default line-ending style for Project Builder editor windows. You can change the default setting in Project Builder by displaying the script file in an editor window, clicking in the script, then choosing Use Mac Line Endings (CR) from the Line Endings submenu of the Format menu.

Even if you do not experience the line endings problem (or you are using Mac OS X version 10.3 or later), you may still be unable to compile the script. A likely cause is that Script Editor doesn't know where to find AppleScript Studio terminology you use in the script. You can tell Script Editor about AppleScript Studio's terminology by enclosing the copied script statements in a Using Terms From

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

block. The block can specify any AppleScript Studio application, such as any of the sample applications you've built. Listing 3-6 shows how to do this by getting terminology from the Drawer application (shown in Figure 1-1 (page 18)).

Listing 3-6 Telling Script Editor where to look for AppleScript Studio terminology

```
using terms from "Drawer"
    (* Insert script statements from your AppleScript Studio
        script file here. *)
end using terms from
```

For more information on working with Xcode's source code editor, see "Source Code Editor" (page 49).

Scripting AppleScript Studio Applications

AppleScript Studio applications use scripts to respond to user actions and perform operations, but are they themselves scriptable applications? The answer is yes, but perhaps not as fully as you might expect.

In script files in the sample applications distributed with AppleScript Studio, you will see many examples of statements that operate on user interface objects, particularly by comparing or setting properties of the objects. For example, the Drawer application gets the state of the "Drawer" button and obtains the "Open drawer on" value (Left, Right, Top, or Bottom) from the matrix of radio buttons, then opens or closes the drawer in the specified location.

You can script similar operations on the running Drawer application from a separate script executed, for example, in Script Editor. The script shown in Listing 3-7 sets a local variable to the current text from the "Date" text field on the application's drawer. To identify the field, it uses a statement similar to ones in the Drawer application script file `Content Controller.applescript`:

Listing 3-7 Setting text in the Drawer application from an external script

```
tell application "Drawer"
    set theText to contents of text field "Date Field" of drawer
        "Drawer" of window "Main"
end tell
```

Note: The line beginning with "set theText" and the following line are all one script statement. As described in "How Xcode Formats Scripts" (page 69), you can wrap text in an Xcode editor window without use of the AppleScript continuation character (¬).

However, there are limitations to this approach. AppleScript Studio makes an application's user interface scriptable—it doesn't make the underlying object model scriptable. For example, if an application window isn't currently open, a script won't be able to access user interface objects in that window. Thus scripts that "script the UI" suffer from the inherent problems of knowing what user interface objects are available when the script runs, as well as how to identify the desired object.

Because of these limitations, turning your Cocoa application into an AppleScript Studio application will not automatically allow robust scripting of the application's features. So using AppleScript Studio for QA testing, for example, is likely to be of most value in simple cases, or for Cocoa applications that already support scripting of their object model.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

For related information, see "Performing User Interface Actions" (page 85), "Experiment With Script Editor to Find Terminology" (page 79), and "Adding AppleScript Studio Support to Your Cocoa Application" (page 86).

Using Script Editor to Test AppleScript Studio Terminology

You can use Script Editor (or a third party script editor) to test the terminology you'll need to access an object in your running AppleScript Studio application. That is, you can target the application, write a statement that gets an object such as the front window, write another statement that gets an object in the front window, and so on. As you execute each script statement to get an object, you can examine the result in the Script Editor's Result pane. By repeating this process, you can identify terminology that will work in your AppleScript Studio script.

Listing 3-8 shows script statements for obtaining a value from a text field in the Drawer sample application. Before using these statements, build and run the Drawer application. Then open a script window in Script Editor (located in `/Applications/AppleScript`) and show the result by clicking the Result tab at the bottom of the window.

Listing 3-8 Scripting the Drawer application from Script Editor

```
tell application "Drawer"
    set theWindow to the front window
    (*if that works, then add the next line *)
    set theTextField to text field "content width" of theWindow
    (* and so on, depending on your object hierarchy *)
    set theText to contents of text field "content width" of theWindow
    display dialog (theText)
    (* and so on *)
end tell
```

To work through this example, start with an empty Tell block:

```
tell application "Drawer"

end tell
```

You can then insert each statement in turn, execute the script, and examine the result. For example, adding the first statement, `set theWindow to the front window` gives you a result something like (`window id 1 of application "Drawer"`) in the Result window. You can then keep adding statements and checking the results until you've figured out how to specify the object, property, or element you're interested in.

In Listing 3-8, the `display dialog` statement merely displays the text extracted from the text field. Starting with AppleScript Studio 1.1, you can obtain a similar result within an AppleScript Studio application script with the `log` command, using statements such as `log theWindow`.

Note: You cannot currently ask for the contents of a reference to a text field—you have to access the text field directly. So replacing the sixth line in Listing 3-8 with `set theText to contents of theTextField` will result in an error.

Listing 3-9 shows a script you can run in Script Editor that obtains a list of all the views in the front window of an AppleScript Studio application named "applicationName". The script logs each view and its class.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Listing 3-9 Examining the views of an AppleScript Studio application

```
tell application "applicationName"
    set viewList to (views of window 1)
    repeat with aView in viewList
        log aView
        log class of aView
    end repeat
end tell
```

If you run this script for the Currency Converter sample application distributed with AppleScript studio (starting with version 1.1), you will get output something like the following in Script Editor's the Event Log pane:

```
(*view id 2*)
(*text field*)
(*view id 3*)
(*text field*)
(*view id 4*)
(*text field*)
(*view id 5*)
(*text field*)
(*view id 6*)
(*text field*)
(*view id 7*)
(*text field*)
(*view id 8*)
(*box*)
(*view id 9*)
(*button*)
```

AppleScript Studio Terminology

The primary documentation for AppleScript Studio terminology is *AppleScript Studio Terminology Reference*. It provides a complete reference to the available terminology, including classes, events, commands, and enumerations.

The following sections describe AppleScript Studio terminology and how it is used:

"Overview" (page 73)
"General Sources of Scripting Terminology" (page 74)
"Terminology From the AppleScriptKit Framework" (page 75)
"Finding Terminology Information" (page 77)

Overview

AppleScript allows you to write scripts that control multiple applications, including many parts of the Mac OS itself. The power in your scripts comes primarily from the scripting terminology provided by the applications and the operating system, not from the relatively small number of terms that are

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

native to AppleScript itself. Scripting additions from Apple and from third parties provide additional terms. (A **scripting addition** is code, stored in Mac OS X in `/System/Library/SystemAdditions`, that makes additional commands or coercions available to scripts on the same computer.)

To take full advantage of the capabilities available, you need to know what terminology you can use in your scripts. Scripts in AppleScript Studio applications have access to the basic terminology that is available to all scripts, as well as to additional terminology that is available to Cocoa applications and, finally, to terminology defined by AppleScript Studio itself. Each of these sources of terminology can include class definitions (which include elements and properties), event and command definitions (which have an associated syntax), and enumerations (or predefined constants).

Note: AppleScript Studio draws a distinction between a command, which is a word or phrase you can use in a script to request an action, and an event, which is an action an object can respond to. That is, scripts can send commands to objects, while events, often the result of user actions, generate calls to event handlers in scripts.

When you examine the terminology for an AppleScript Studio application, you'll see both commands and events listed together in the "Commands" section. For more information, see "Terminology Browser" (page 51).

General Sources of Scripting Terminology

Sources of terminology that are not unique to AppleScript Studio include:

- terminology provided by AppleScript
- terminology from scriptable parts of the Mac OS
- terminology from available Apple and third party scripting additions
- terminology from available scriptable applications (whether Carbon or Cocoa applications)
    - Cocoa applications store scripting terminology in a **script suite**, described in "Terminology From the AppleScriptKit Framework" (page 75).
    - Carbon applications store scripting terminology in an `'aete'` resource, described in the AppleScript documentation listed in "See Also" (page 15).
    - You can examine the terminology for either type of application, as described in "Terminology Browser" (page 51).

Note: The name spaces of these various terminologies sometimes conflict, which can result in confusing scripting errors.

Cocoa applications have access to scripting information derived from the script suites of the application itself (including Cocoa's default suites), any scriptable frameworks the application uses, and any scriptable bundles it loads. An AppleScript Studio application has access to the default terminology that is available to it as a Cocoa application, as well as to terminology it defines in its own framework, which gives the application the ability to script Cocoa user interface objects.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Terminology From the AppleScriptKit Framework

AppleScript Studio applications are built with the Cocoa application framework and provide terminology to allow scripters to make use of Cocoa application, document, user interface, and other objects. That terminology comes from two sources: terms that are available to all Cocoa applications that support scripting, and terms defined by AppleScript Studio's AppleScriptKit framework.

Terms From Cocoa's Built-in Suites

Cocoa frameworks and applications provide scripting information in the form of one or more script suites. A **script suite** consists of at least one suite definition and one suite terminology, contained in external files. A **suite definition** describes scriptable objects in terms of their attributes, relationships, and supported commands. This information is stored as key-value pairs (where each pair has an identifying key and a corresponding value) in a property list. A **property list** is a structured, textual representation of data, commonly stored in Extensible Markup Language (XML) format.

A **suite terminology** provides corresponding AppleScript terminology—the English-like words and phrases you can use in a script—for the class and command descriptions in a suite definition. Suite terminologies are also stored as property lists. Frameworks and applications typically place terminology files in a localized resource directory named `English.lproj`. (English is currently the only supported dialect in AppleScript.)

AppleScript Studio applications can take advantage of the built-in Cocoa terminology found in two default suites, the Standard and Text suites.

- The Standard suite:
  - Defines the Abstract Object class. This class serves as a parent class for all other classes. It has just one property, the Class of the object.
  - Defines basic classes, including Application, Document, and Window (though you'll see in the next section that AppleScript Studio defines its own version of these classes in its Application and Document suites).
  - Defines terminology for basic events, including Get, Set, Count, Delete, Print, Quit, and others. In Cocoa applications that turn on scripting support (as previously described in "Terms from The AppleScriptKit Framework" (page 75)), objects can support certain key events, such as Get and Set, with little or no extra code.
- The Text suite defines classes for working with text, such as Character, Paragraph, Word, and Text.

For more information on Cocoa's built-in scripting support, see the documentation described in "See Also" (page 15).

Terms from The AppleScriptKit Framework

AppleScript Studio adds to the two default suites defined by Cocoa so that it can provide additional terminology for scripting Cocoa's many user interface objects. This terminology is defined in several suite terminology files in AppleScript Studio's own framework, the AppleScriptKit framework. For detailed information on the classes and events in these suites, see "Finding Terminology Information" (page 77).

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

The **Application suite** defines its own version of some common classes that are defined in the Standard suite (described previously), including Application and Window classes. It also defines the Item class, which has Name and ID properties, and the Responder class, which inherits from the Item class and serves as a superclass for the Window, View, and Control classes. These and other classes that inherit from Responder can respond to user actions.

Note: In AppleScript Studio 1.2, the document class is described in a separate Document suite and a Drag and Drop suite has been added.

To work with the many high-level classes it contains, the Application suite defines a large number of events for working with the application, windows, mouse and keyboard events, and so on. Prior to AppleScript Studio 1.2, the Application suite defined the Document class. That class is now defined in its own suite.

The **Container View suite** defines the View class, as well as additional classes whose primary purpose is to contain other views. These include Box, Clip View, Drawer, Scroll View, Split View, Tab View, and Tab View Item. Except for Tab View Item, all of the classes in the Container View suite inherit from Responder, either directly or through the View class. The Container View suite also defines events for working with container views and the views they contain.

The **Control View suite** defines a number of classes for implementing or working with controls, including Button, Cell, Color Well, Control, Image View, Movie View, Popup Button, Progress Indicator, Slider, and Text Field. Controls are graphic objects that cause instant actions or visible results when the user manipulates them with the mouse. The Cell class inherits from the Abstract Object class (described in "Terms From Cocoa's Built-in Suites" (page 75)), while other classes in this suite inherit from the View class, either directly or through the Control class.

The Control View suite defines many events for working with user actions involving controls.

The **Data View suite** defines classes whose primary purpose is to display rows and columns of data. These include Browser, Browser Cell, Data Cell, Data Column, Data Row, Data Source, Outline View, Table Column, Table Header Cell, Table Header View, and Table View. You've worked with several of these classes in building the Mail Search application. The classes in the Data View suite generally inherit from either the View class, the Cell class, or the Abstract Object class (described in "Terms From Cocoa's Built-in Suites" (page 75)).

The Data View suite defines events for working with the items, cells, rows, and columns found in table and outline views.

The **Document suite** defines AppleScript Studio's version of the document class defined in the Standard suite (described previously). This suite defines terminology you can use to work with documents in any AppleScript Studio application, but which is most important to document-based applications. The Document suite first became available in AppleScript Studio 1.2. In prior versions, a smaller Document class was defined as part of the Application suite.

The **Drag and Drop suite** defines terms for working with drag and drop, including the `drag info` class to provide information about a drag, and events to drag, track, prepare for a drop, and conclude a drop. The Drag and Drop suite first became available in AppleScript Studio 1.2.

The **Menu suite** is a small suite that defines just two classes and two events for working with menus. The classes are Menu and Menu Item, which both inherit from the Abstract Object class (described in "Terms From Cocoa's Built-in Suites" (page 75)). The events are Choose Menu Item and Update Menu Item.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

The **Panel suite** defines classes and events for dealing with dialogs, alerts, and panels. The classes include Alert Reply, Color Panel, Dialog Reply, Font Panel, Open Panel, Panel, and Save Panel. The two reply classes inherit from the Abstract Object class (described in "Terms From Cocoa's Built-in Suites" (page 75)), while the other classes inherit from Window, either directly or through the Panel class.

The **Plugin suite** defines terms for working with application plug-ins. Starting in AppleScript Studio version 1.3, first distributed with Mac OS X version 10.3, Xcode provides a new template for creating AppleScript plug-ins for Xcode. That is, you can use AppleScript Studio to create a plug-in that adds features to Xcode itself. The Plugin suite provides terminology to use with plug-ins of this type. For more information, see "AppleScript Studio Xcode Plug-in Template" (page 43).

The **Text View suite** defines two classes for displaying and manipulating text: Text and Text View. Text inherits from View and Text View inherits from Text.

Finding Terminology Information

The primary documentation for AppleScript Studio terminology is *AppleScript Studio Terminology Reference*, available in the AppleScript Documentation area. This section describes several additional ways to obtain information about the scripting terminology available to AppleScript Studio applications.

Examine Scripting Dictionaries

You can get detailed information about the currently available terminology by examining AppleScript Studio's scripting terminology in a dictionary viewer, as described in "Terminology Browser" (page 51). You can open any of the AppleScript Studio sample projects in Xcode and select `AppleScriptKit.asdictionary` in the Files list of the Groups & Files list. It provides a link to a scripting terminology file in the AppleScriptKit framework. By displaying this file in a dictionary viewer (also known as a terminology browser), you can examine the terms that are available to all Cocoa applications that support scripting (in the Standard and Text suites) and those that are specific to AppleScript Studio applications (the Application, Container View, Control View, Data View, Document, Drag and Drop, Menu, Panel, and Text View suites).

Figure 3-3 shows the open dictionary in an Xcode window; the classes in the Data View suite are visible, with the Data Source object selected.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Figure 3-3 The AppleScript Studio scripting dictionary in Xcode

Each class description in the dictionary shows the plural form for the class (if applicable), its elements (if any), and its properties (if any). Every class except the Abstract object class (described in "Terms From Cocoa's Built-in Suites" (page 75)) has an inheritance property, and through it inherits the properties of its superclasses. A property is labeled "[r/o]" if it is read only (you can't set its value).

Each command description in the dictionary shows the syntax for invoking the command. Parameters enclosed in brackets, such as `[on]`, are optional.

Investigate the Sample Applications

The sample applications distributed with AppleScript Studio provide valuable examples of the terminology for working with objects based on most AppleScript Studio classes. Each of the sample applications demonstrates a small number of features, so that you can more easily focus on the details. These applications are described in "AppleScript Studio Sample Applications" (page 35).

Table 3-3 lists some of the sample applications and the key objects used in each application. You can examine the application script files to find examples of terminology for working with these types of objects.

Table 3-3 AppleScript Studio sample applications and the objects they use

Application	Objects used
Archive Maker	button, checkbox, custom view, panel, progress indicator, text field, text view; uses `call method` command to call Objective-C methods; uses `do shell script` command
Assistant	box, button, image view, tab view, text field
Browser	browser, browser cell, column, Finder items, browser row

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Application	Objects used
Currency Converter	box, button, label text, number formatter, text field
Display Alert	alert, button, checkbox, matrix (radio buttons), text field, window
Display Dialog	dialog, button, checkbox, text field, window
Display Panel	dialog, button, checkbox, matrix (radio buttons), panel, text field, window
Drag Race	button, dialog, progress indicator, slider, text field, window
Drawer	button, drawer, matrix (radio buttons), panel, text field, steppers, window; uses `call method` command to call Objective-C methods
Image	image, image view
Language Translator	box, button, popup button (pop-up menu), progress indicator, slider, split view, text field, window; makes SOAP calls
Mail Search (formerly Watson)	button, outline view, progress indicator, scroller, split view, table view, window
Multi-Language	button, menu, menu item, text field, window
Open Panel	button, open-panel, text field, window
Outline	child, Finder items, outline view, row, table column
Save Panel	button, save-panel, text field, window
Simple Shell	text field, text view, window; uses `do shell script` command
SOAP Talk	button, progress indicator, scroll view, text field, window; makes SOAP calls
Table	button, column, data cell, data source, row, table view, text field
Talking Head	bundle, frame, menu item, movie, movie view, window
Unit Converter	box, button, popup button (pop-up menu), text field, window; uses `load script` command

Experiment With Script Editor to Find Terminology

To determine script terminology by targeting an AppleScript Studio application from Script Editor, see "Using Script Editor to Test AppleScript Studio Terminology" (page 72).

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Programming Tips

The following sections provide information that may be useful as you build applications with AppleScript Studio:

- "Targeting an AppleScript Studio Application" (page 80)
- "Using Make, Not Create, to Create New Objects in Scripts" (page 80)
- "Using the Log Command to Track Your Scripts" (page 80)
- "Basic Tips and Reminders" (page 81)

Targeting an AppleScript Studio Application

You do not have to use a `tell application` statement in an AppleScript Studio application script because scripts implicitly target the application itself.

Using Make, Not Create, to Create New Objects in Scripts

Scripters are familiar with using AppleScript's `make` command to create new objects. To make a new object, you specify the class to make, the location, and optionally the properties and data for the object. For example, to make a new "name" data column in a table view, the Table sample application uses the following statement:

```
make new data column at the end of the data columns with properties {name:"name"}
```

For more information on tables and data columns, see the Table application, or see "Controller Script Properties and Initialization" (page 193) in the Mail Search tutorial. For more information on the `make` command, see AppleScript Language Guide.

Using the Log Command to Track Your Scripts

During development, you can insert `log` statements in your AppleScript Studio scripts to help keep track of what's going on. The `log` command outputs a value to the console view of Xcode's "Run" pane if you run the application from Xcode, or to the Console application (located in `/Applications/Utilities`) if you run the application from the Finder.

You can log references to objects or strings or variables. The following examples show `log` statements and their results:

```
log "Testing" -- "Testing"
log theObject -- "view id 23 of view id 10 of window id 1"
```


2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Basic Tips and Reminders

This section provides a number of tips and reminders that may improve your experience in working with AppleScript Studio.

Comment AppleScript Studio Handlers

Comments are generally a good thing, and in AppleScript Studio it's particularly useful to comment event handlers. For example, Listing 3-10 shows the code AppleScript Studio inserts in your script when you attach a `clicked` handler to a button (assuming you haven't already created a `clicked` handler in the same script).

Listing 3-10 A new clicked handler

```
on clicked theObject
    (* Add your script here. *)
end clicked
```

There are several things you might do to help make this code more self-explanatory:

- If the handler is called for one of several types of control objects, you can add a comment to that effect:

  ```
  on clicked theObject
      (* This handler handles controls on the Search Parameters pane. *)
      -- Your script statements here
  end clicked
  ```

- If you know the `clicked` handler will only be called for one object (say a "Search" button), you can change the name of the `theObject` parameter accordingly (to, for example, `theSearchButton`). Changing the name has no effect on how the handler operates.
- If the handler may called for one of several objects (say a series of buttons), you can both change the parameter name and add a comment:

  ```
  on clicked importButton
      (* This handler handles buttons on the Import pane. *)
      -- Your script statements here
  end clicked
  ```

  For an example that shows how to distinguish between multiple named buttons in a handler, see Listing 3-1 (page 62).

Save Your Work

Some things are obvious but still need repeating. Any time you've done significant work on the code or interface for your AppleScript Studio application, save your work. Even though Mac OS X is very robust. Even if you have an interruptible power supply. And while you're at it, occasionally save your entire project to another drive. (Pardon the lecture.)

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Occasionally Do a Clean Rebuild

You should occasionally use Xcode's "Clean active target" button (or the Clean or Clean All Targets items in the Build menu) to do a full rebuild of your application. Cleaning removes all derived products and files (such as `.o` files). Doing a clean build can sometimes eliminate odd results when running or debugging the application.

Give All Important Objects an AppleScript Name

When you're adding objects to your nib file in Interface Builder, consider giving an AppleScript name to any objects you may want to script in the future, even if you don't currently plan to script them. That way you're prepared when you're working on a handler and you realize you need to access a particular object.

You can enhance the benefits of this approach by using a consistent naming convention. For example, you can always name your main window "main" and come up with a standard way of naming buttons, text fields, and so on. Then when you need to access one of those items in a script, it will be easy to remember their names. It may also help eliminate a common bug—spelling an AppleScript name differently in a script than in Interface Builder.

Troubleshooting

This following sections provide tips that may be useful in troubleshooting your AppleScript Studio applications.

- "My Script Statements Aren't Working" (page 82)
- "Several Windows in My Application Have ID 0" (page 83)
- "I Can't Script My UI to Do QA Testing" (page 83)

My Script Statements Aren't Working

If your script compiles but doesn't work in the running application, there are several options for determining the problem:

- The runtime error message may point you at the problem. Some runtime messages are notoriously unhelpful—efforts are being made to improve them.
- You can use Script Editor (or a third-party script editor) to test the terminology you're using in your scripts. For details, see "Using Script Editor to Test AppleScript Studio Terminology" (page 72).
- You can use the `log` command to get more information, as described in "Using the Log Command to Track Your Scripts" (page 80).
- You can use the standard basic debugging tactics of inserting `beep` or `display dialog` commands in your scripts.
- You may be able to use powerful third-party debuggers with AppleScript Studio. Details will be provided as soon as they are available.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Several Windows in My Application Have ID 0

When you add new windows to a nib file in Interface Builder, they all have ID 0. Interface Builder doesn't attempt to assign serial index numbers to windows (though it does for buttons or other items you place in a window).

If you so desire, you will still be able to access windows by index in your application, with statements such as

```
set myButton to the first button of the second window
```

I Can't Script My UI to Do QA Testing

This topic is described in "Scripting AppleScript Studio Applications" (page 71).

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

AppleScript Studio Cookbook

This chapter provides step-by-step instructions for performing common AppleScript Studio tasks, in the following sections:

- "Performing User Interface Actions" (page 85)
- "Specifying Minimum Requirements for an Application" (page 86)
- "Adding AppleScript Studio Support to Your Cocoa Application" (page 86)
- "Setting the Keyboard Focus" (page 87)
- "Obtaining the Path to the Current Application" (page 87)

Performing User Interface Actions

AppleScript Studio provides the ability to perform user interface actions directly in scripts, using the `perform action` command (defined in the Control View suite). For example, you can tell an interface object, such as a button, to perform its `clicked` handler, thus providing a way to directly script the user interface (subject to limitations described in "Scripting AppleScript Studio Applications" (page 71)). Note, however, that calling the `clicked` handler will not provide the visual feedback a user would see if they actually clicked the button.

Listing 4-1 shows a script that tells the "Drawer" button in the Drawer application to perform its `clicked` handler, which will either open or close the drawer, depending on its current state.

Listing 4-1 Manipulating a button in the Drawer application from an external script

```
tell application "Drawer"
    set theButton to button "Drawer" of window "Main"
    tell theButton to perform action
end tell
```

The `perform action` command does nothing unless the specified object has an action handler—a handler such as a `clicked` or `double-clicked` handler in the Action group in the Interface Builder Info window for the object. For example, see the Info window in Figure 1-15 (page 32).

To use the `perform action` command with menus, you can use syntax like the following:

```
tell menu item 1 of menu 1 of main menu to perform action
```


2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

You can also call application methods directly, as described in "Scripting AppleScript Studio Applications" (page 71).

Specifying Minimum Requirements for an Application

To run AppleScript Studio applications, the target machine must include the AppleScript Studio runtime required for the application. The runtime is available if `AppleScriptKit.framework` is present in `/System/Library/Frameworks`.

For example, an application built with AppleScript Studio 1.2 that uses features added in version 1.2 requires the 1.2 runtime. However, a similar application that doesn't use any features from AppleScript Studio 1.2 can run with the 1.1 runtime. Note that all 1.1 applications can run with the 1.0 runtime distributed with Mac OS X version 10.1.2. For more information on versions and runtimes, see "AppleScript Studio System Requirements and Version Information" (page 221).

You can install a `will finish launching` handler for your application to check the version. "Connect the Application Object" (page 169) shows how to add a `will finish launching` handler.

Listing 4-2 shows a simple example of how the `will finish launching` handler might check for the application's required version of AppleScript Studio. In this case, the application quits if the required version isn't available. Note that the handler doesn't check AppleScript Studio's version number directly. Instead, it checks for the corresponding AppleScript version, as shown in Table A-1 (page 221).

Listing 4-2 will finish launching handler that checks for required version of AppleScript Studio

```
on will finish launching theObject
    considering numeric strings
    if AppleScript's version as string is less than "1.10.1" then
        display dialog "This application requires AppleScript Studio 1.4 or later."
        quit
    end if
    end considering
end will finish launching
```

Adding AppleScript Studio Support to Your Cocoa Application

You can use the following steps to add AppleScript Studio support to your existing Cocoa application:

1. Add the AppleScriptKit framework to your project. You can do so by navigating to `/System/Library/Frameworks/AppleScriptKit.framework` and dragging the framework into the Frameworks group in the Xcode project for your application.

   In the dialog that appears, do not select the checkbox to copy items, but do check the radio button to recursively create groups. If you have more than one target, you'll have to select any targets the framework should be added to.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

2. Add an AppleScript Studio build phase to the application. To do this, first click the Targets tab, then double-click the desired Target. You should see a pane showing the Files & Build Phases tab.

   Click the section labeled Bundle Resources. You should be able to select the whole section.

   Choose Project > New Build Phase > New AppleScript Build Phase (from the Projects menu).

3. If you've already added any `.applescript` files to your application, they'll be in the Bundle Resources phase, and you'll have to drag them to the new AppleScript phase you just created.

4. If your application is not already scriptable, make the following modification to its `Info.plist` file to make it scriptable:

   In the Application Settings pane (after clicking the Expert button), add a new string entry to the property list that sets NSAppleScriptEnabled to "YES". You can add an entry by clicking the New Sibling button. You can double-click any of the entries in the sibling to edit it.

5. If your application has a `.scriptsuite` file, for every class in that file whose superclass belongs to the core suite (NSCoreSuite), change the suite to ASKApplicationSuite. For example, change

   ```
   "Superclass" = "NSCoreSuite.NSApplication"
   ```

   to

   ```
   "Superclass" = "ASKApplicationSuite.NSApplication"
   ```

Setting the Keyboard Focus

To make a user interface object, such as a text field, have the keyboard focus, set its window's `first responder` property. For example, the following Tell statement causes a text field to become active and be first in line to respond to keyboard events:

```
tell window "user information"
    set first responder to text field "user name"
end tell
```

Note that without the Tell statement, you would have to specify the window twice:

```
set first responder of window "user information"
    to text field "user name" of window "user information"
```

In many cases, you will not have to set keyboard focus, because it is set automatically as a user tabs between fields, clicks on a text field, and so on.

Obtaining the Path to the Current Application

To obtain the path to the current (running) AppleScript Studio application, you use the standard scripting addition command `path to`. For example:

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

```
set myPath to (path to me)
display dialog (myPath as string)
-- result, if inserted in on opened handler in Drawer sample application:
-- MacOSX:Developer:Examples:AppleScript Studio:Drawer:build:Drawer.app
```

The Standard Additions are automatically installed with Mac OS X.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Currency Converter Tutorial

In this chapter, you'll create a simple AppleScript Studio application that converts a dollar amount to an amount in another currency. The Currency Converter tutorial describes a number of tasks that are common to most AppleScript Studio applications, including:

- creating a project with Xcode
- building an interface with Interface Builder, including
  - inserting and initializing user interface objects
  - adjusting the interface to comply with the Aqua guidelines
  - connecting the interface to an event handler in a script
  - using common shortcuts
- writing an event handler
- building and running the application

You can also find a plain Cocoa version of the Currency Converter Tutorial in the Objective-C Language Documentation area of the Cocoa Documentation. That tutorial provides some information you won't need in this chapter, such as a discussion of object-oriented design, steps for creating custom object classes, and steps for hooking up user actions to class methods. In the AppleScript Studio version of Currency Converter, you'll handle user actions in a scripting event handler, rather than with custom classes and methods. That makes your task quite a bit simpler. However, you may want to consult the Cocoa tutorial before designing more complex AppleScript Studio applications, such as the one described in "Mail Search Tutorial: Design the Application" (page 119).

To build the Currency Converter application, you'll perform these steps:

1. "Design the Application" (page 90)
2. "Create a Project" (page 90).
3. "Build the Interface" (page 91).
4. "Connect the Interface" (page 112)
5. "Write Event Handlers" (page 115)
6. "Build and Run the Application" (page 117)

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

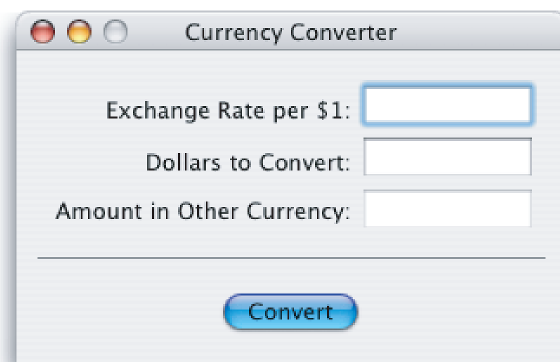
This chapter assumes you have completed the Hello World tutorial in "Creating a Hello World Application" (page 27) and read "AppleScript Studio Components" (page 39) as well.

After completing the Currency Converter tutorial, see "Where To Go From Here" (page 117) for suggestions on how to learn more about AppleScript Studio.

Design the Application

The Currency Converter application is simple enough that it doesn't require a complicated design process. The application should enable a user to type in a conversion rate and a dollar amount, then click a button to see the result—how much the dollars are worth in the new currency. The application requires only one window, which can be configured as shown in Figure 5-1.

Figure 5-1 The Currency Converter window



A user enters values in the first two fields, then clicks the button to get the answer in the third field. A horizontal separator divides the input and display fields from the Convert button.

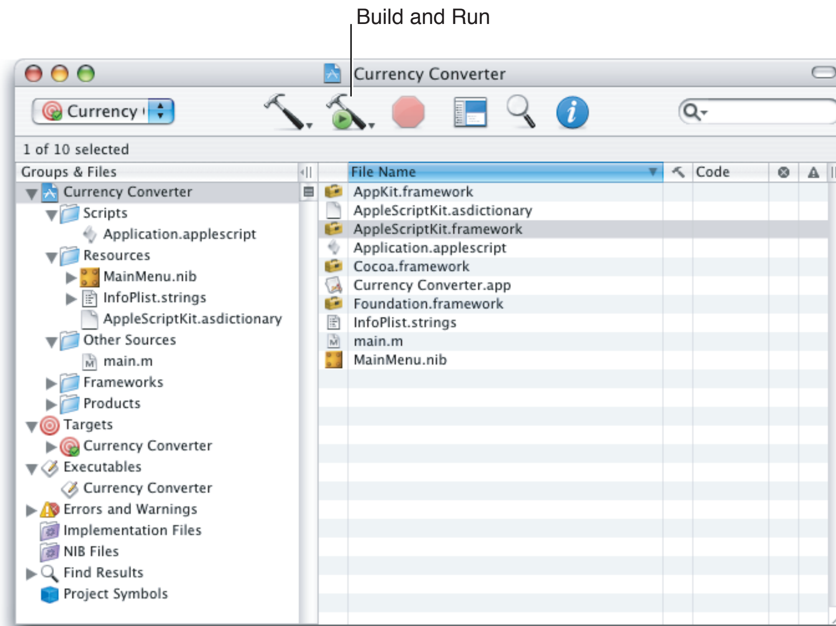
Create a Project

Use Xcode to create a new AppleScript Studio project, as described in "Creating a Hello World Application" (page 27), typing "Currency Converter" for the project name.

Figure 5-2 shows the new project after opening several groups in the Files list in the Groups & Files pane. You can save the project by typing Command-S or by choosing the Save command from the File menu. Use your typical level of care in periodically saving and backing up your work.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Figure 5-2 The project after opening several groups



At this point, you can build the project and create a working application, with a window that can be expanded, minimized, and closed. The application can display an About window and respond to a number of menu choices. These features are provided by the Cocoa application framework (described in "Cocoa Framework Overview" (page 58)), without further work on your part. Of course you've got some work to do before the application can perform its custom operation of converting currencies.

To build and run the application, do one of the following:

- type Command-R
- choose Build and Run from the Build menu
- click the Build and Run button shown in Figure 5-2

When you are ready to continue with the tutorial, quit the application.

It usually makes sense to build an application periodically to make sure you haven't introduced errors. However, the Currency Converter application is fairly simple, and it won't have any additional functionality until you've written an event handler, so you might want to wait until later in this tutorial.

Build the Interface

Before working on this section, you should be familiar with the information in "Interface Builder Features for AppleScript Studio" (page 53), which describes, among other things, the nib files Interface Builder uses to store interface definitions.

To build the interface for Currency Converter, you perform these steps:

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

1. "Launch Interface Builder" (page 92)
2. "Adjust the Title, Size, and Other Attributes of the Currency Converter Window" (page 92)
3. "Add Text Input Fields and Labels" (page 98)
4. "Add a Result Field and Label" (page 105)
5. "Add Number Formatters to the Input and Result Fields" (page 107)
6. "Add a Convert Button" (page 110)
7. "Add a Horizontal Separator" (page 111)
8. "Finalize the Layout" (page 112)

Launch Interface Builder

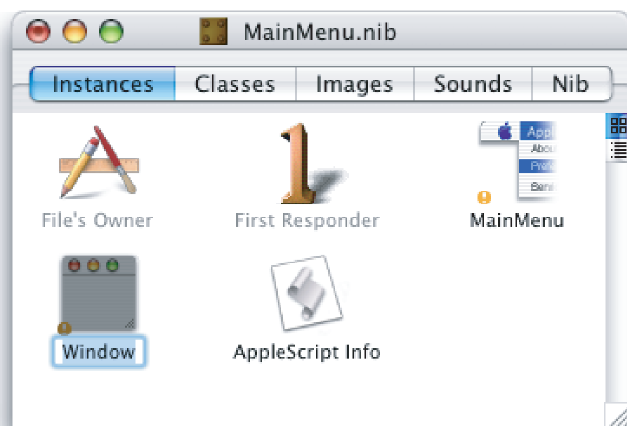
When you create a project with Xcode, the project automatically contains a default nib file named `MainMenu.nib`. The icon for this nib file is visible in Figure 5-2 (page 91). To launch Interface Builder, double-click the icon. You should see the same four windows shown in Figure 2-7 (page 54).

Adjust the Title, Size, and Other Attributes of the Currency Converter Window

In this section you'll make changes to the default window to set its title and adjust its size and other attributes. To modify the default window, perform these steps:

1. In Interface Builder's MainMenu.nib window, double-click the text "Window" in the title of the Window instance. The result is shown in Figure 5-3.

Figure 5-3 Selected text for Window instance in MainMenu.nib window

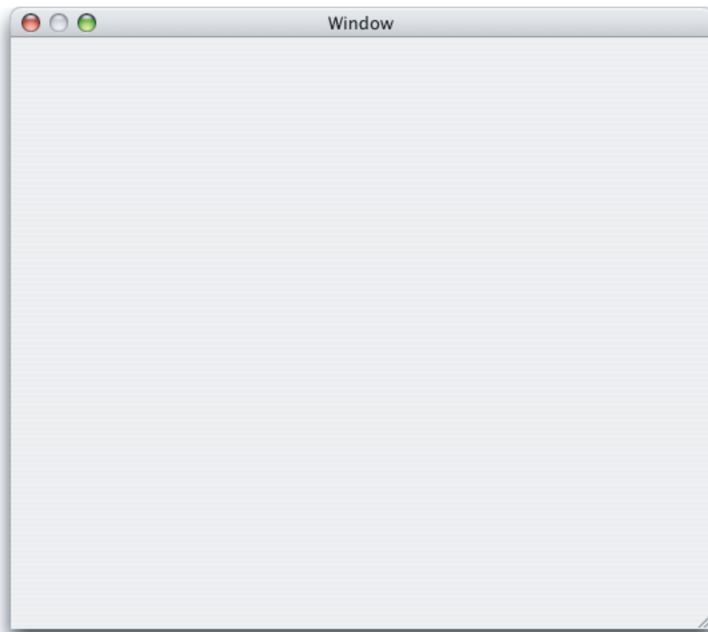


2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

2. Type “Currency Converter” (without the quotes) as the new instance name. This step changes only the instance name, not the window title, but it’s a good habit to always name window instances so you can easily identify them.

3. Figure 5-4 shows the default window for the Currency Converter instance. Note “Window” is still the window title.

Figure 5-4 The default window in the Currency Converter project



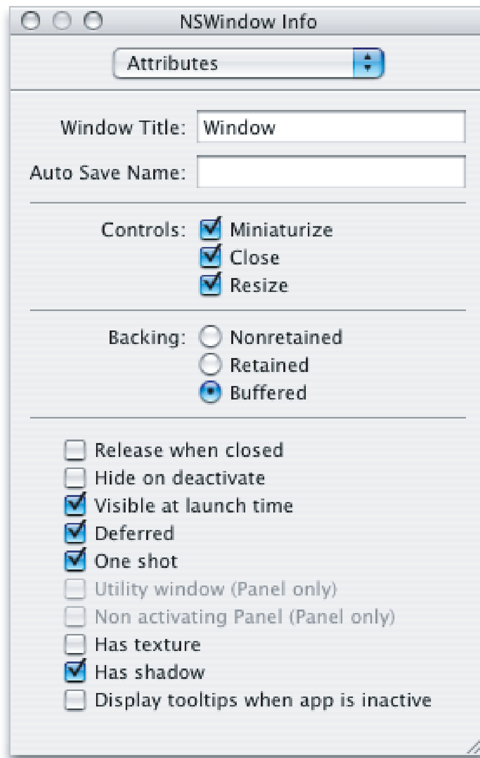
Click once in this window, then choose Show Info from the Tools menu or type Command-Shift-I to open the Info window. The resulting Info window, with the Attribute pane visible, is shown in Figure 5-5.

The Info window displays information about the currently selected object. The Info window displays the Attributes pane by default, or whichever pane was visible the last time the window was open. If the Attributes pane is not visible, use the pop-up menu at the top of the Info window or type Command-1 to display it.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

You can tell the Info window is displaying information for a window object because its title is "NSWindow Info" (the window object is an instance of Cocoa's NSWindow class). If the Info window has some other title when you open it, click either the window instance shown in Figure 5-3 or the window object shown in Figure 5-4 to display window information in the Info window.

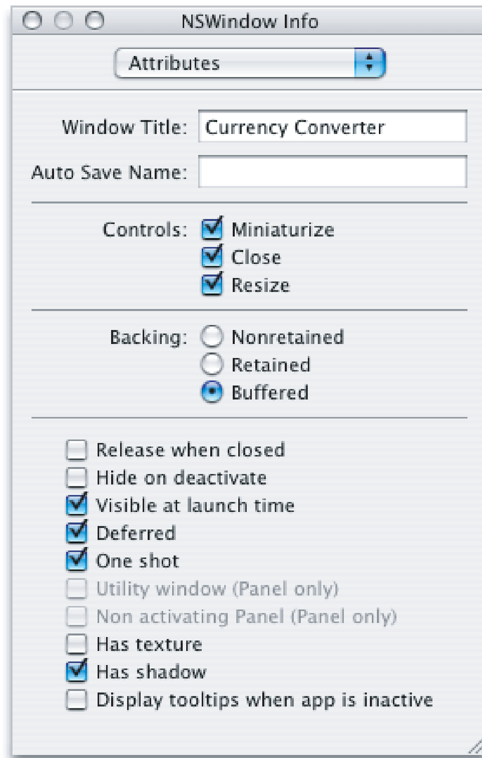
Figure 5-5 The Attributes pane of the Info window for a window object



2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

4. To retitle the window, simply type "Currency Converter" in the Window Title field, as shown in Figure 5-6.

Figure 5-6 The Info window after retitling the Currency Converter window



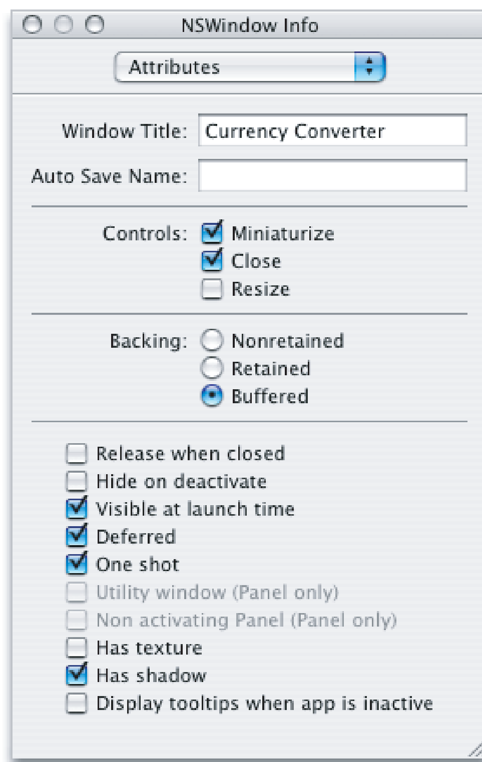
5. In the Controls section, deselect the Resize checkbox—you won't allow a user to resize Currency Converter's window. Later you'll add a handler to quit the application when a user closes the window.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Note: Even though you disable resizing of the window, the Currency Converter window in Interface Builder still has a resize control so you can resize it while working on your interface. When you build and run the application, the window will not have a resize control.

You don't need to change the default settings for any other attributes. The final settings are shown in Figure 5-7. For information on the other attributes, see "AppleScript Studio Terminology" (page 73), as well as Interface Builder Help or the Cocoa documentation for the NSWindow class.

Figure 5-7 The final Attributes settings for the Currency Converter window

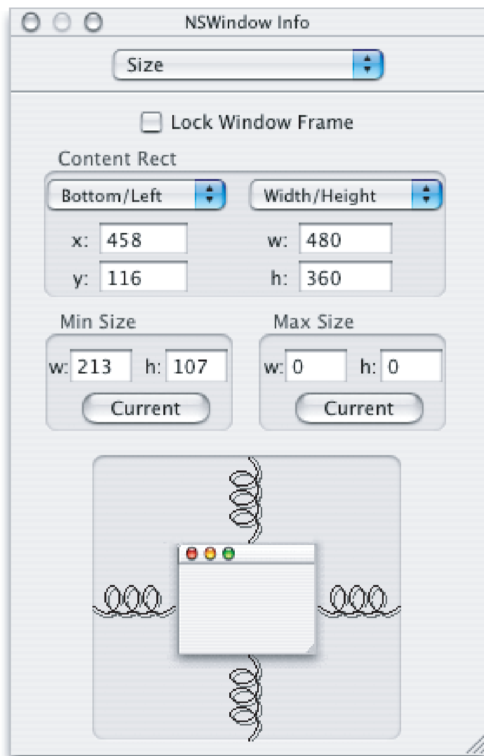


6. To resize the Currency Converter window, drag the resize control in the lower-right corner. The resized window should look similar to the one shown in Figure 5-1 (page 90).

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Instead of dragging, you can set the window to a specific pixel size. Use the pop-up menu or type Command-3 to display the Size pane in the Info window. The result is shown in Figure 5-8.

Figure 5-8 The Size pane of the Currency Converter window



To change the size, enter new width and height values in the text fields below the Width/Height pop-up menu. For the window shown in Figure 5-1 (page 90), the width is about 320 pixels and the height about 180 pixels.

The Bottom/Left pop-up menu determines the window's onscreen position when a user runs the application. You can either drag the window to the desired position or modify the values in the Size pane.

There are several items in the Size pane whose default values don't need to be changed for Currency Converter. For example, the spring images control resizing, but you've already turned off resizing for the Currency Converter window.

After all the changes you've made in this section, the Currency Converter window should look like Figure 5-9.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Figure 5-9 The modified Currency Converter window

Add Text Input Fields and Labels

In this section you'll perform steps that are common to nearly all AppleScript Studio application development, including:

- dragging user interface objects into your application window
- positioning and resizing objects to support the Aqua user interface guidelines
- using Interface Builder's Info window to set attributes and prepare objects for scripting

Currency Converter needs text fields for entering the exchange rate per dollar and the number of dollars to convert. Each of these input fields needs a label.

To add text input fields and labels to Currency Converter's main window, perform these steps:

1. Click the Cocoa-Text button in the Palette window toolbar. The Cocoa-Text palette is shown in Figure 5-10.

   To display a help tag describing a button (and the palette it selects), position the cursor over an item in the Palette window's toolbar for a few seconds.

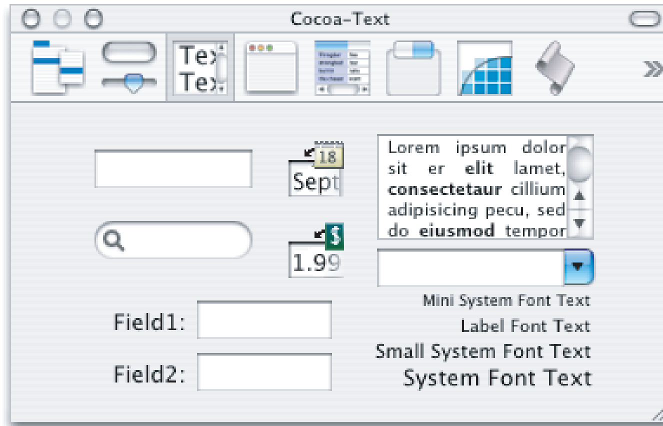
   Similarly, position the cursor over a user interface object in a palette to display the object's Cocoa class (such as NSButton or NSTextField).

   You can see that the palette provides several kinds of text elements. Each text element has a distinct appearance based on its function and role in an application—you can see a search field, a pop-up menu, a wrapping text field, and more.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

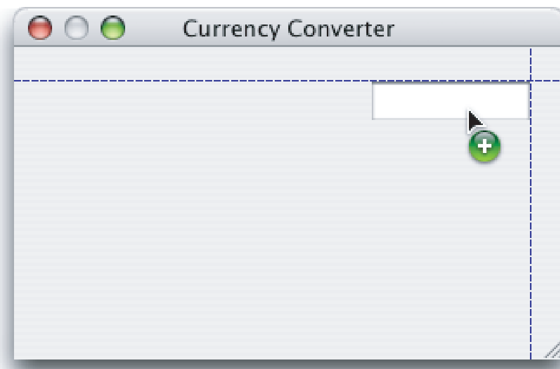
> **Note:** You can modify the Palette window toolbar in Interface Builder by choosing Tools>Palettes>Customize Toolbar. If, for example, you do not see the AppleScript button (the rightmost button in Figure 5-10) in the toolbar, you can add it by customizing the toolbar.

Figure 5-10 The Cocoa-Text palette of Interface Builder's Palette window



2. Drag a text field object from the Palette window to the Currency Converter window. Position the text field in the upper-right corner, using Interface Builder's feedback to help you align the text field according to the Aqua guidelines, as shown in Figure 5-11. This is the exchange rate input field.

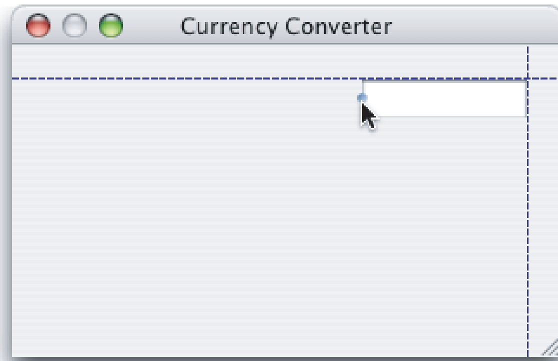
Figure 5-11 Positioning a text input field for the exchange rate



2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

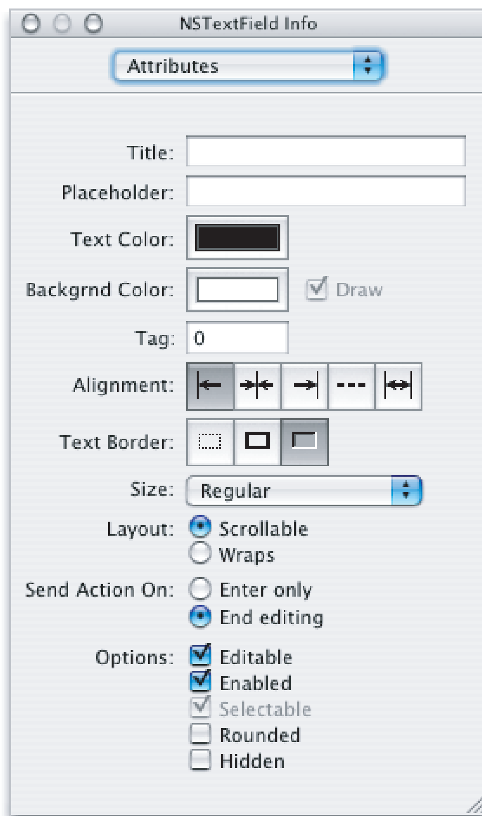
3. Select the text field, then drag the middle selection handle on the left side to resize the field, as shown in Figure 5-12. (If you use the upper or lower handle, it's harder to maintain the same vertical height as you resize the field.) For now, don't worry about making the field any specific width—you can adjust it later if necessary.

Figure 5-12 Resizing the exchange rate input field



4. With the exchange rate field selected, open the Info window to the Attribute pane. The result is shown in Figure 5-13.

Figure 5-13 The Attributes pane of the Info window for the exchange rate field



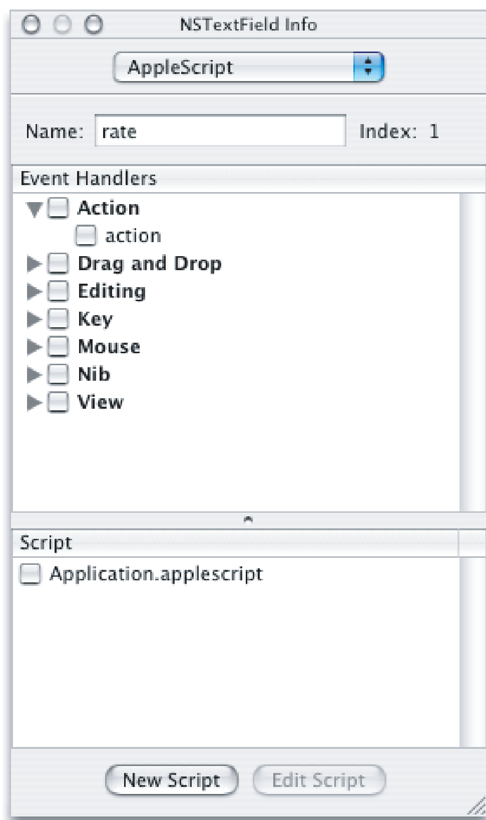
2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

5. In the alignment section, click the middle button, to right-align text in the field.

   There are several attributes whose default values don't need to be changed for Currency Converter. For example, Editable and Enabled are already selected in the Options section. For information on other attributes, see Interface Builder Help or the Cocoa documentation for the NSTextField class.

6. To provide an AppleScript name for the exchange rate input field (so you can access it in scripts), display the AppleScript pane in the Info window, then type "rate" in the Name field. Figure 5-14 shows the result.

Figure 5-14 The Info window, after supplying an AppleScript name for the exchange rate field

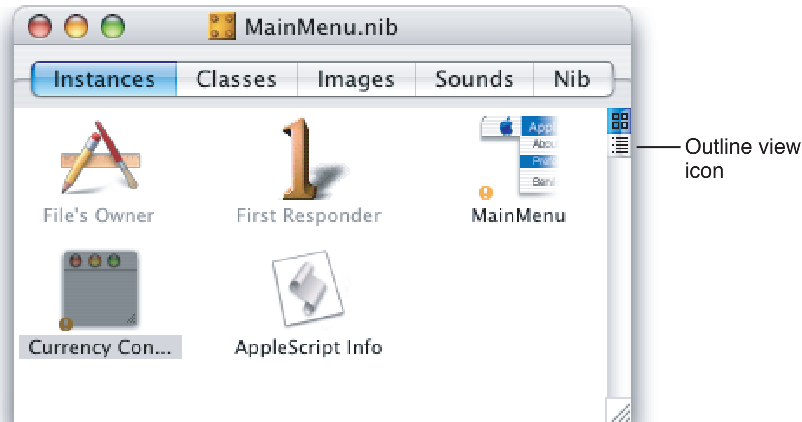


If it makes sense, you can use the same AppleScript name for objects of different types in the same window (such as a button and a text field), or for objects of the same type (such as buttons) in different windows. However, if you name two text fields in the same window "rate," you won't be able to differentiate between them by name in an application script.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

> **Note:** The first time you make a change in the AppleScript pane, Interface Builder adds a new instance named AppleScript Info to the nib window. This instance is shown in Figure 5-15. For more information about this object, see "Add the Message Window to the Nib File" (page 139).

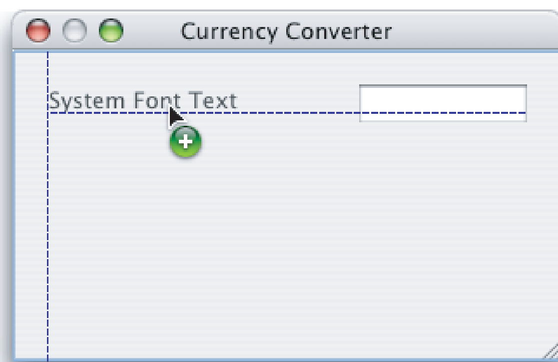
Figure 5-15 The MainMenu.nib window showing an AppleScript Info object (not selected)



> **Note:** Figure 5-15 shows the Instances pane of the MainMenu.nib window in icon view. You can instead view it in outline view, by clicking the small outline view icon on the right-hand side of the view. An example of outline view is shown in Figure 8-12 (page 183).
>
> Outline view is convenient for displaying view hierarchies and connections, and for selecting deeply nested objects. Later in this tutorial, you may want to use outline view to examine Currency Converter's object hierarchy after inserting number formatters (see "Add Number Formatters to the Input and Result Fields" (page 107)).

7. To add a label for the exchange rate input field, drag the label text field object with the text "System Font Text" (shown in Figure 5-10 (page 99)) from the Palette window to the Currency Converter window. Use the automatic feedback to position the label field to the left of the exchange rate input field, as shown in Figure 5-16.

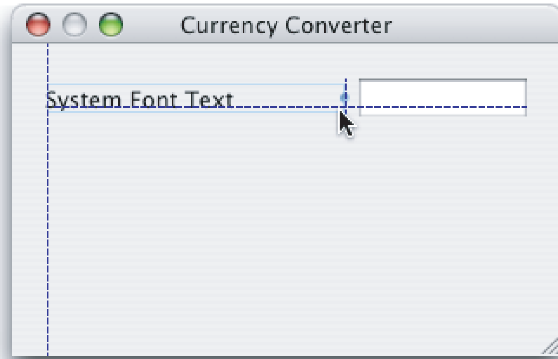
Figure 5-16 Positioning a label field for the exchange rate



2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

8. As in a previous step, select the label field and drag the middle selection handle on the left side to resize the field, as shown in Figure 5-17.

Figure 5-17 Resizing the label field for the exchange rate



9. With the label field selected, open the Info window and display the Attributes pane. Now make these adjustments:

   a. Type "Exchange Rate per $1:" in the Title field. This is the label text. If any of the text is not visible, it may have scrolled out of view. If so, widen the label field until all the text is visible. You may need to resize the exchange rate input field as well.

   b. In the Alignment section, click the middle button, to right-align the label text.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Figure 5-18 shows the resulting Info window; Figure 5-19 shows the text label field. Note that a label field, like an input field, is based on the NSTextField class.

Figure 5-18 The Info window, after setting text and attributes for the exchange rate label

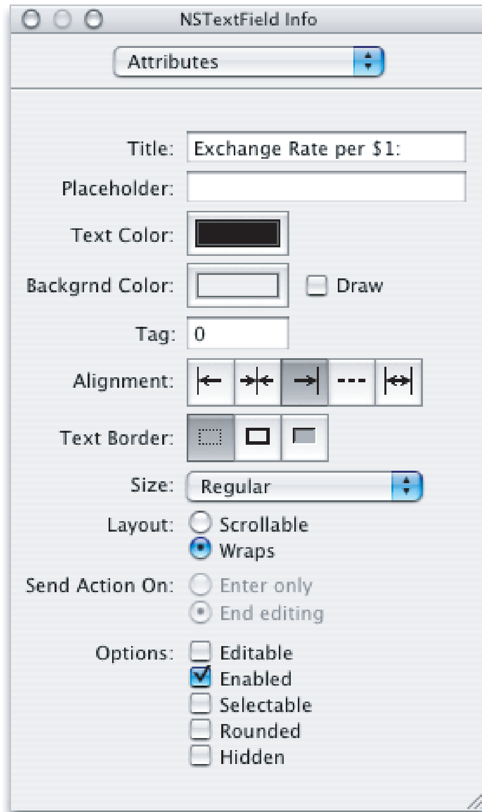
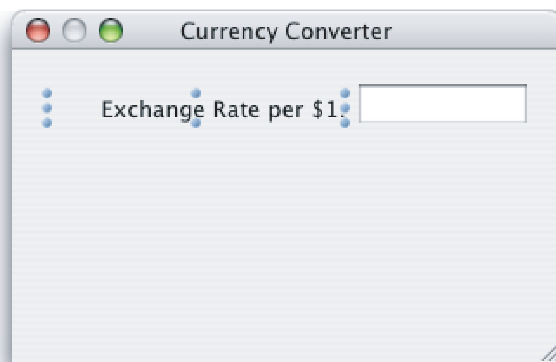


Figure 5-19 The exchange rate label field (selected)



10. Save all the changes you've made in Interface Builder—they're saved to the `MainMenu.nib` file. You'll also be given a chance to save your changes if you quit Interface Builder.

11. To add the text input and label fields for "Dollars to Convert," you can repeat your previous steps:

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

a. Drag a text field into the Currency Converter window, aligning it below the exchange rate text field.

b. Resize the field.

c. Set its attributes.

d. Give it the AppleScript name “amount”.

e. Add a label field, aligning it below the previous label field.

f. Label the field “Dollars to Convert:” and right-align it.

Or, you can use Interface Builder’s Duplicate command to duplicate the two already-created text fields, then supply them with new label text and new AppleScript names:

a. Shift-click or drag to select the two fields you’ve already entered, then choose Duplicate from the File menu or type Command-D.

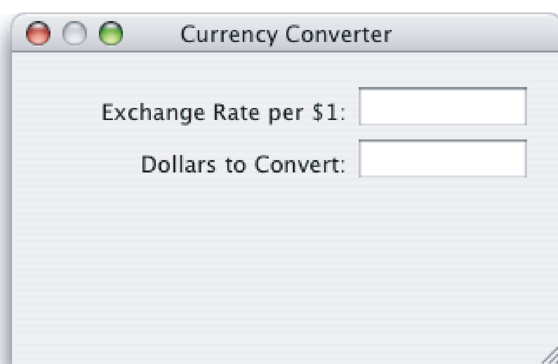
b. Drag the duplicated fields, using the automatic guides to align them below the original fields.

c. Select the new label field and in the Attributes pane of the Info window, type “Dollars to Convert:” in the Title field.

d. To provide an AppleScript name for the text input field, select the field, open the Info window to the AppleScript pane, and type “amount” in the Name field.

Figure 5-20 shows the results of adding the two new text fields.

Figure 5-20 The Currency Converter window with input fields and labels



Add a Result Field and Label

In this section you’ll add a text field to display the computed total in the new currency. You’ll also add a label field for the conversion result. You’ll get additional tips on aligning fields in “Finalize the Layout” (page 112).

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

1. To add a text field and label for "Amount in Other Currency," perform these steps:

   a. Shift-click or drag to select the exchange rate label and text input fields, then choose Duplicate from the File menu or type Command-D.

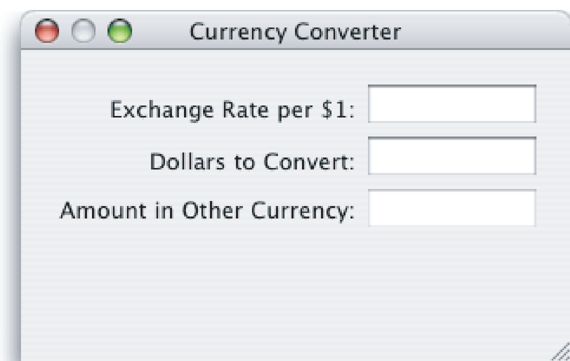
   b. Drag the duplicated fields and align them below the original fields using the guides.

   c. Select the new label field and type "Amount in Other Currency:" in the Title field in the Attributes pane of the Info window.

   d. Select the text input field and type "total" in the Name field in the AppleScript pane of the Info window.

   Figure 5-21 shows the results of adding the additional text fields. The window now contains its final collection of text fields and labels.

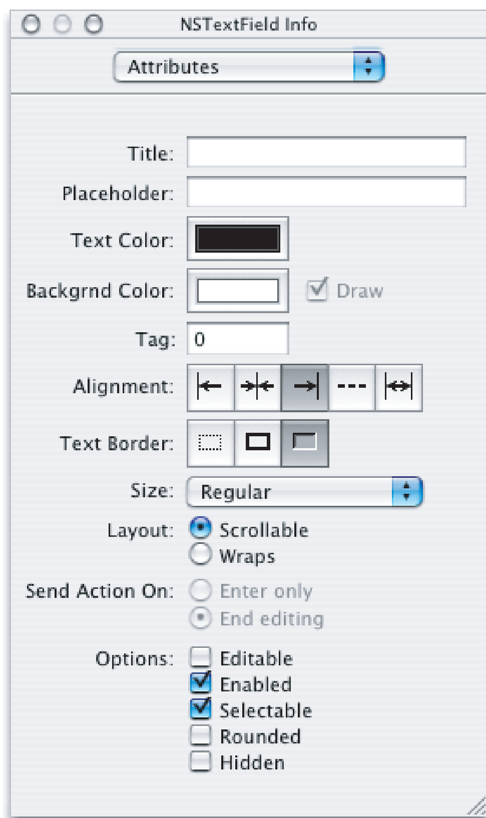
Figure 5-21 The Currency Converter window with all text fields and labels



2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

2. You don't want a user to be able to enter text in the "Amount in Other Currency" input field, so you'll need to disable editing. Select the field, display the Attributes pane in the Info window, and deselect the Editable checkbox in the Options section. Figure 5-22 shows the result.

Figure 5-22 The Info window, after disabling editing for the amount in other currency field



Save your work before moving on to the next section.

Add Number Formatters to the Input and Result Fields

Currency Converter should allow users to input only meaningful values, and it should display a properly formatted result:

- the exchange rate input field should accept and display decimal numbers such as 1.55 or 10.24
- the dollars to convert input field should display valid dollar amounts, such as $125.20
- the amount in other currency field should display the computed total in the appropriate currency to the current locale, such as dollars ($600.10) or yen (¥1700.00)

To format the text in a text field, Interface Builder provides number formatters (based on the Cocoa NSNumberFormatter class). In the Cocoa-Text palette in Figure 5-10 (page 99), the number formatter is represented by a dollar sign superimposed on the number 1.99.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

> **Important:** When a number formatter is attached to a text field, the contents must be explicitly a number, as shown in Listing 5-2 (page 116), and not as text.

To provide number formatters for Currency Converter's text fields, perform these steps

1. Select the exchange rate input field in the Currency Converter window.
2. Open the Info window to the Attributes pane.
3. Display the Cocoa-Text palette in Interface Builder's Palette window.
4. Drag a number formatter from the palette and drop it on the input field, as shown in Figure 5-23.

Figure 5-23 Adding a number formatter to the exchange rate input field

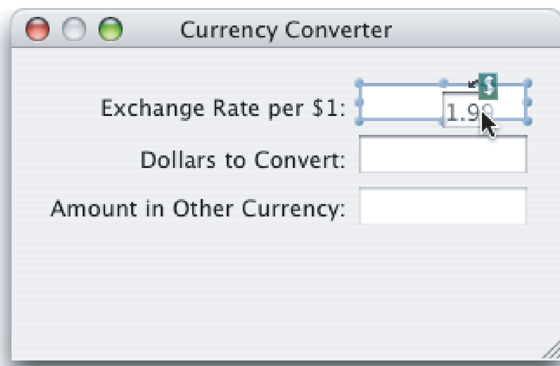
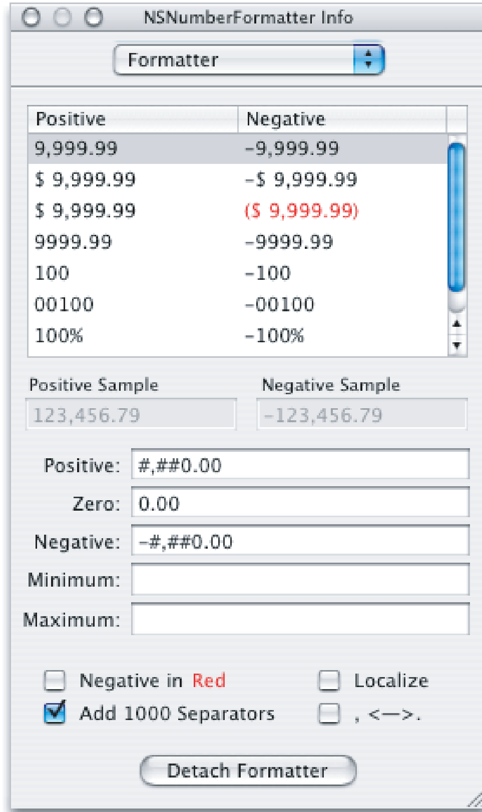


Figure 5-24 shows Info window for the number formatter. The formatter pane should open automatically when you insert a number formatter into a text field. You can also select the pane from the pop-up menu.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

The default formatter settings are correct for the exchange rate input field, so you don't need to make any changes.

Figure 5-24 The Formatter pane for the exchange rate input field



> **Note:** See the Note with Figure 5-15 (page 102) for information on viewing Currency Converter's MainMenu.nib window in outline view. In outline view, you can examine the application's object hierarchy, and see each number formatter object and the text field it is associated with.

5. Repeat the previous steps to add a number formatter to the dollars to convert input field. The Info window should again look as in Figure 5-24.

6. Select the second row in the formatter, so that the field includes a dollar sign.

7. Repeat the previous steps to add a number formatter to the amount in other currency field. The Info window should again look as in Figure 5-24. Again, select the first format row that includes a dollar sign.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

8. Because the amount in other currency is likely not to be in dollars, select the Localize checkbox in the Options section. Figure 5-25 shows the result.

Figure 5-25 The Formatter pane for the amount in other currency field

For more information on number formatters, see Interface Builder Help or the Cocoa documentation for the NSNumberFormatter class.

Add a Convert Button

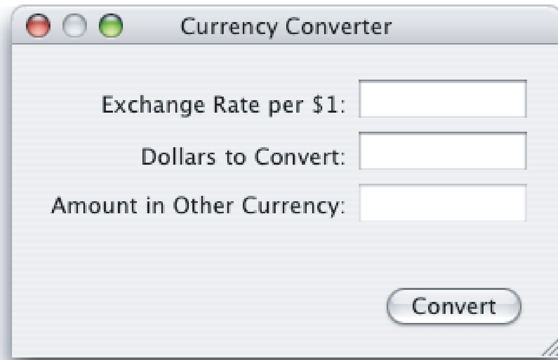
To add a Convert button to Currency Converter, perform the following steps:

1. Drag a button object from the Cocoa-Controls palette in the Palette window to the bottom right corner of the Currency Converter window. Use the automatic feedback to align the right edge of the button with the right edge of the text field above it.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

2. Double-click the button to select its text, then type “Convert”. Figure 5-26 shows the result.

Figure 5-26 The Currency Converter window with a “Convert” button



Add a Horizontal Separator

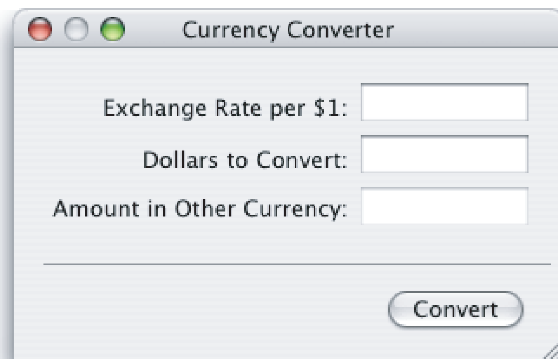
To add a horizontal separator to Currency Converter, perform the following steps:

1. Drag a box object from the Cocoa-Controls palette in the Palette window to the Currency Converter window. The box object looks like a horizontal line. (If you let the cursor rest over the line in the Palette window, Interface Builder displays “NSBox”.)

   Position the box in the lower right of the window, between the convert button and the text field above it, with the right edge near the right edge of the window. Use the automatic feedback to align it.

2. Drag the left selection handle to resize the separator, using the automatic feedback to stretch it until it almost reaches the left side of the window. Figure 5-27 shows the result.

Figure 5-27 The Currency Converter window with a horizontal separator



2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Finalize the Layout

In this tutorial, you have generally positioned user interface objects so that they are aligned with other objects and in accord with the Aqua interface guidelines. However, now that all the user interface objects are in place, you can check the alignment and perform other layout operations.

1. Click and shift-click to select all three of the label fields.
2. To resize the fields to their minimum sizes, choose Size to Fit from the Layout menu.
3. To right align the labels, choose Align Right Edges under Alignment in the Layout menu.
4. Similarly, select all three non-label text fields.
5. To make them the same size, choose Same Size from the Layout menu.
6. To align their left edges, choose Align Left Edges under Alignment in the Layout menu.
7. To make sure they're still aligned according to the Aqua guidelines, use the automatic feedback as you drag them to the right edge of the window.

See Interface Builder Help for information on other options in the Layout menu. And remember, you can use these layout options at any time that makes sense.

Connect the Interface

The Currency Converter application needs a connection so that when a user clicks the Convert button (or presses the Return key), the application calls a handler in an application script. It also needs a connection so that when a user closes the window, the application quits (the application can't do anything useful without a window).

To set up connections for Currency Converter, perform these steps in Interface Builder:

1. Select the Convert button, then display the Attributes pane in the Info window. Figure 5-28 shows the row in the middle of the Attributes pane that you use to select a key equivalent for the button.

   Figure 5-28 Fields from the Info window for setting keystroke equivalent

   In the pop-up that says <no key>, choose Return. This causes the Convert button's action handler (the `clicked` handler you'll connect next) to be called when a user presses the Return key. Figure 5-29 shows the result.

   Figure 5-29 Fields from the Info window for setting keystroke equivalent

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

When you choose the Return key as the keystroke equivalent for a button, you've designated that button as the default button in the window. Cocoa automatically colors the button aqua to show it is the default button.

You can use the checkboxes to the right of the pop-up to include the Option and Command keys as part of the keystroke equivalent, but that isn't necessary for Currency Converter.

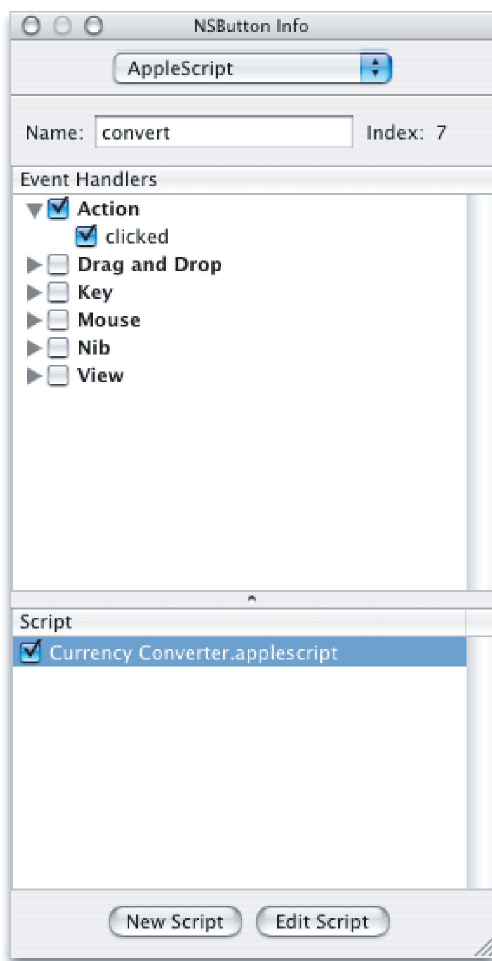
2. With the Convert button still selected, display the AppleScript pane in the Info window.

3. Type "convert" for the AppleScript name for the button.

4. Click the disclosure triangle next to the Action checkbox and select the "clicked" checkbox to indicate the application should have a `clicked` event handler, called when a user clicks the button.

5. Check the file `Currency Converter.applescript` in the Script list to connect the handler. Figure 5-30 shows the Info window after this step.

Figure 5-30 The Info window after connecting a clicked handler to the Convert button



2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

6. Save your results. That causes Interface Builder to insert an empty `clicked` handler in the selected script file. If you click the Edit Script button, Interface Builder also opens the file in an Xcode editor window. You'll see the code for the `clicked` handler in the next section. But for now, just save your results and stay in Interface Builder so you can add another handler.

7. To quit the application when a user closes the Currency Converter window, you'll need to connect a special handler to the application. Figure 5-3 (page 92) shows the MainMenu.nib window. The File's Owner instance in that window represents NSApp, a global constant for the application object that serves as the master controller for the application. (The icons in the Instances pane are described in "Interface Creation" (page 54).)

   Select the File's Owner in the nib window and open the Info window to display the AppleScript pane.

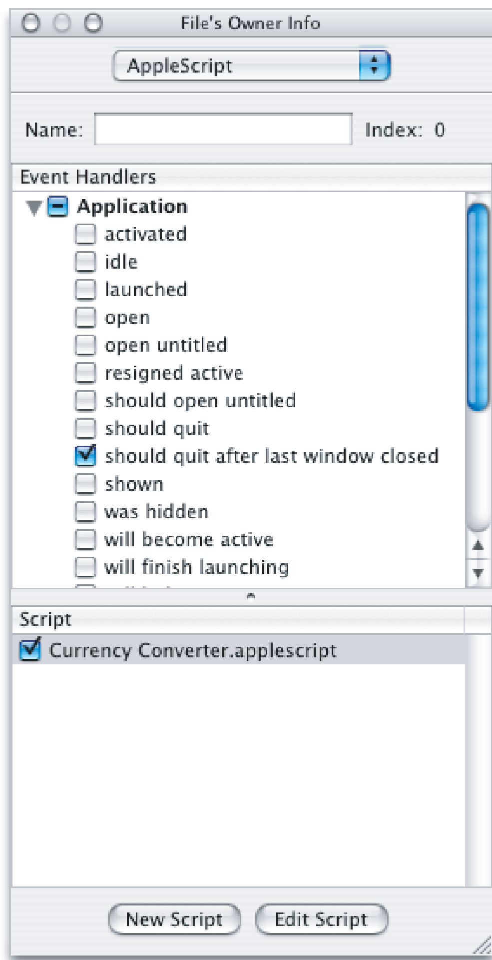
8. Click the disclosure triangle next to the Application checkbox and select the "should quit after last window closed" checkbox. Clicking this checkbox indicates the application should have a corresponding event handler, called when a user closes the last window (the Currency Converter's only window).

   The dash in the Application checkbox indicates that at least one handler in the group is checked, but not all are.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

9. Select the file `Currency Converter.applescript` in the Script list to connect the handler. Figure 5-31 shows the Info window after this step.

Figure 5-31 The Info window after connecting a handler to the application object



Write Event Handlers

When you connect a handler in Interface Builder and save your changes, Interface Builder inserts a skeleton handler into the specified script in your Xcode project. Listing 5-1 shows the skeleton `clicked` handler in the file `Currency Converter.applescript`. The application calls this handler when a user clicks the Convert button. The parameter `theObject` is a reference to the button object itself.

Listing 5-1 The empty clicked handler

```
on clicked theObject
    (* Add your script here. *)
end clicked
```


2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

In the `clicked` handler, the Currency Converter application needs to get the exchange rate and the number of dollars to convert, multiply them to determine the total in the new currency, and display that value. It should behave gracefully if the value for either the exchange rate or the number of dollars is missing. (The number formatters you added to the input fields prevent a user from saving erroneous values, such as alphabetic characters.)

Listing 5-2 shows the completed `clicked` handler.

Important: You do not have to use a `tell application "Currency Converter"` statement in an application script because scripts implicitly target the application itself. For that reason, the `clicked` handler in Listing 5-2 contains no `tell application` statement, though it does use a `tell window` statement to target the window on which it accesses various text fields.

Listing 5-2 The complete clicked handler

```
on clicked theObject
    tell window of theObject
        try
            set theRate to contents of text field "rate" as number
            set theAmount to contents of text field "amount" as number
            set contents of text field "total" to theRate * theAmount
        on error
            set contents of text field "total" to 0
        end try
    end tell
end clicked
```

The key points of this handler are:

- the `theObject` parameter is a reference to the clicked button object
- a button, like other controls, is a view, and a script can get the window of a view
- a script can get a reference to any named AppleScript object within a window (it can also refer to items by number—for example, `the first text field`— or ID, but names are easy to specify, self-documenting, and won't change dynamically)
- the handler shows the syntax for getting and setting the content of text fields; when a number formatter is attached to a text field, the contents must be set explicitly as number, as shown in Listing 5-2, and not as text
- the actual work performed by the Currency Converter application is quite simple
- the handler uses a `try`, `on error` block to set the contents of the result field to 0 if an error occurs in calculating the converted value; this protects the user from receiving a possibly confusing error message if they attempt to convert without supplying both an exchange rate and an amount to convert

When you connect a handler in Interface Builder and click the Edit Script button, Interface Builder inserts a skeleton handler into the specified script and opens the script in Xcode. Listing 5-3 shows the completed `should quit after last window closed` handler in the file `Currency Converter.applescript`. This handler is called when a user closes the Currency Converter window. Its one line, supplied by you, returns the value `true`, indicating the application should, indeed, quit when the last window is closed.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Listing 5-3 The should quit after last widow closed handler

```
on should quit after last window closed theObject
    return true
end should quit after last window closed
```

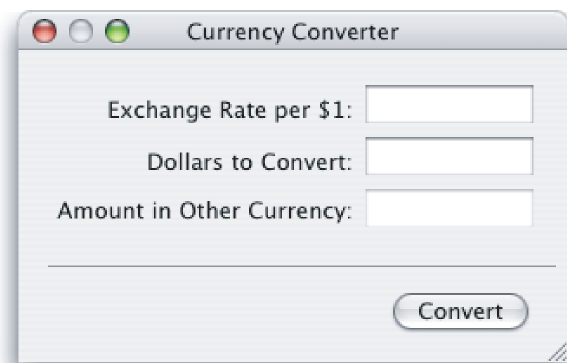
Build and Run the Application

The Currency Converter application should now be ready to perform conversions, so it's time to build it again. Use any of the mechanisms described previously to build and run the application:

- type Command-R
- choose Build and Run from the Build menu
- click the Build and Run button (a combined monitor and hammer)

Figure 5-32 shows the application after computing a currency conversion. Because the test computer was set up for U.S. English, the converted amount is shown in dollars, but for another locale it might show yen or pounds.

Figure 5-32 The Currency Converter application in action



For a detailed discussion of building and checking for syntax errors, see "Mail Search Tutorial: Build and Test the Application" (page 205).

Where To Go From Here

Once you have completed the Currency Converter tutorial, you have several options for learning more about AppleScript Studio:

- If you haven't already done so, build and experiment with some of the sample applications described in "AppleScript Studio Sample Applications" (page 35).

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

- If you're ready for a more challenging tutorial, go right to the chapter "Mail Search Tutorial: Design the Application" (page 119).
- If you want to take it a little slower, try using the information in "Chapter 12, Mail Search Tutorial: Customize the Application," (page 209) to customize the Currency Converter application. Although the customizing chapter refers frequently to Mail Search, the customization steps can be applied to any application.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Mail Search Tutorial: Design the Application

In this chapter you'll use AppleScript Studio to design a fairly complex AppleScript Studio application called Mail Search. Mail Search searches for text in messages in the Mac OS X Mail application. You'll design Mail Search by performing the steps described in the following sections:

> **Note:** The Mail Search sample application distributed with AppleScript Studio was formerly known as "Watson".

1. "Identify a Goal for the Application" (page 120).
2. "Examine Mail's Scripting Dictionary" (page 120).
3. "Specify Operations for Mail Search" (page 122).
4. "Design the Interface" (page 123).
5. "Plan the Code" (page 127).

You'll complete the Mail Search application in the following chapters:

"Mail Search Tutorial: Create the Interface" (page 133)
"Mail Search Tutorial: Connect the Interface" (page 169)
"Mail Search Tutorial: Write the Code" (page 187)
"Mail Search Tutorial: Build and Test the Application" (page 205)
"Mail Search Tutorial: Customize the Application" (page 209)

This organization demonstrates a design decision to use separate steps to create the interface, connect it, and write scripts to perform operations. Working in this manner is appropriate for a tutorial, where the outcome is known in advance, but once you're familiar with AppleScript Studio, it may not be the most convenient way to design your own applications. Instead, you may choose to work more incrementally, adding part of the interface, connecting it to a handler, and testing (even if only with diagnostic statements to show that program flow is working properly). This tutorial points out places where you might choose to build and test the application in the normal course of development.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Before You Start This Tutorial

Before starting the Mail Search tutorial, it is recommended that you complete the "Currency Converter Tutorial" (page 89), read "AppleScript Studio Components" (page 39), and experiment with some of the sample applications described in "AppleScript Studio Sample Applications" (page 35). In particular, the Table and Outline sample applications demonstrate how to work with table view and outline view objects.

Identify a Goal for the Application

If you're a regular user of the Mac OS X Mail application, you may know that it has a convenient feature for finding specified text in the messages within a mailbox, but prior to Mac OS X version 10.2, it could only search one mailbox at a time. Suppose you recently got mail from a friend and couldn't remember if you filed it in Personal Mail, To Do List, Read Later, or some other mailbox. You could look for the message in any single mailbox by searching for your friend's name, or for a word or phrase you remember from the message. You could specify what part of the message to search—Subject, To, From, or Entire message text—but there was no option to search more than one mailbox at a time.

Well, many users had to wait for Mac OS X version 10.2 to be able to search across mailboxes. But AppleScript Studio supplied this feature in Mac OS X version 10.1, through the Mail Search application, one of the sample applications distributed with AppleScript Studio. And though this feature is now built in to Mail, the Mail Search application still provides a useful introduction to a full-featured AppleScript Studio application.

In this tutorial, you'll perform all the steps required to design and build the Mail Search application. In the process, you'll gain experience with many AppleScript Studio features, including

- working with document-based applications
- creating and connecting complex user interface objects, using multiple nib files
- writing scripts to control outline and table views, progress bars, and other interface objects
- incorporating script objects in an application
- building AppleScript Studio applications

While designing and building the Mail Search application may prove a challenging task, by the time you complete the tutorial you should have a good grasp of many of the key tools and concepts needed to build AppleScript Studio applications. And even if you don't work through every step of the tutorial, you can browse it to find tips for performing specific tasks.

Examine Mail's Scripting Dictionary

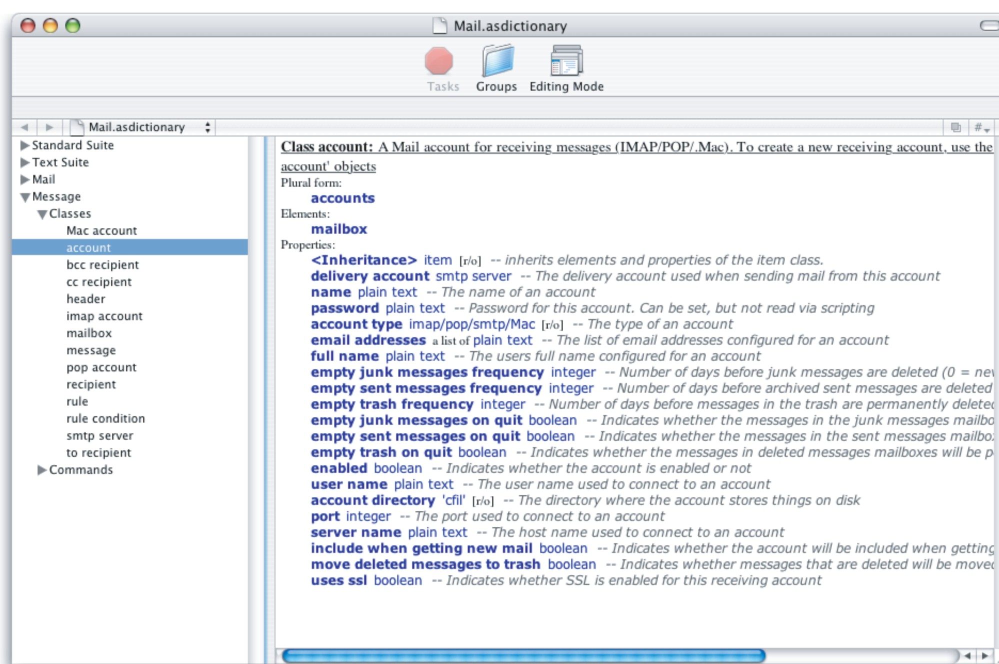
Before you can design an AppleScript Studio application to script the Mail application, you need to know what scripting terminology Mail supports. To examine Mail's scripting dictionary, do the following:

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

1. Open Xcode.
2. Choose Open Dictionary from the File menu.
3. Choose the Mail application. If it doesn't show up among the available scriptable applications, you can navigate to it by clicking the Browse button. The application itself is located in `/Applications`.

You'll find more information on examining scripting dictionaries in "Terminology Browser" (page 51). Figure 6-1 shows Mail's scripting dictionary in an Xcode pane. (You can also open dictionaries in the Script Editor application. Both applications provide the same information.)

Figure 6-1 The Mail application's scripting dictionary in an Xcode window



Note: The scripting terminology for the Mail application changed with Mac OS X version 10.2. The scripts and examples created prior to that version differ from their counterparts today. The listings shown in this document reflect terminology revisions through Mac OS X version 10.3.

Figure 6-1 shows several terminology suites, or collections of related classes and events. The Standard Suite and Text Suite are default suites that all Cocoa applications get just by turning on scripting (as described in "Cocoa Scripting Support" (page 58)). The Mail and Message suites are specific to Mail. Take some time to examine the classes and events available in each suite. Mail Search is likely to require scripting terms from the Mail and Message suites, as well as terms from the Standard suite, such as `get` and `set`.

For example, to search through messages in all mailboxes, Mail Search has to access:

- accounts (elements of the application class in the Mail suite)

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

- the mailboxes they contain (elements of the account class)
- the messages (elements of the mailbox class) in those mailboxes

Look for other classes whose properties and elements Mail Search might need, as well as for events that may be useful. As it turns out, Mail Search doesn't require any events from the Message or Mail suites, but does use events from the Standard Suite (which includes such built-in AppleScript commands as `get` and `set`). In fact, Mail Search uses the `get` command extensively to get accounts, mailboxes, and messages.

Note: See "Terminology Browser" (page 51) for a description of how AppleScript Studio distinguishes between events and commands.

Specify Operations for Mail Search

The goal of the Mail Search application is to search multiple Mail application mailboxes for specified text. A Mail user can have multiple mail accounts—for example, an IMAP account for work-related email, a POP account from an Internet service provider, and perhaps additional accounts as well. Each account can have multiple mailboxes, including mailboxes within other mailboxes.

A user should be able to search all or a selected group of mailboxes and to specify which part of the messages to search: Subject, To, From, or Contents (in Mail, the equivalent of Contents is "Entire message text"). For lengthy operations, Mail Search should display a progress bar. On completion of a search, Mail Search should display a list of messages that contain the specified text. Columns in the list should display the From and Subject fields, as well as the name of the mailbox the message is in.

Finally, a user must be able to read individual messages that contain the search text. Because Mail's scripting support doesn't currently support opening an individual message in a Mail window, Mail Search can instead display a message in a separate window it creates.

Based on this analysis, Mail Search's requirements can be summarized as follows:

- Mail Search can obtain and display a list of all mailboxes from all accounts.
- A user can specify which mailboxes to search, by specifying one or more of the available mailboxes.
- A user can specify text to search for.
- A user can specify where to search from the following choices:
  - Subject field
  - To field
  - From field
  - Contents of message
- Mail Search can find and display all messages containing the specified text. The result list includes
  - From
  - Subject
  - Mailbox name

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

- A user can double-click a message in the result list to open it in a separate window.
- During long operations, such as searching through multiple mailboxes and potentially thousands of messages, Mail Search can provide feedback on its progress.

As you'll see in later sections, the Cocoa application framework and the Cocoa user interface objects you use to implement Mail Search have many built-in features. As a result, Mail Search automatically gains many capabilities not listed here.

Design the Interface

One way to design an interface is to perform these simple steps:

1. Figure out what the application does so that you can describe what information and user actions the interface must be able to handle.

   This step has already been completed, in "Specify Operations for Mail Search" (page 122).

2. Identify the kinds of user interface widgets you might use to implement the interface.

   AppleScript Studio supplies plenty of widgets—namely all of Cocoa's user interface objects. This step is described in "Identify Objects for the User Interface" (page 123).

3. Arrange the widgets in a pleasing format. This step is described in "Arrange the User Interface" (page 125).

Identify Objects for the User Interface

Before specifying requirements for Mail Search, which works closely with the Mail application, you investigated the scriptable features supported by Mail. Similarly, before designing Mail Search's interface, you should investigate the user interface objects available to AppleScript Studio applications. "Cocoa User Interface Objects" (page 59) describes how to view user interface objects in the Palettes window in Interface Builder. You can also take a look various sample applications distributed with AppleScript Studio, including Browser and Outline, which demonstrate the use of column and outline views that imitate the Finder's column and list views. For more information on the sample applications, see "AppleScript Studio Sample Applications" (page 35).

To help narrow the search for objects, here are Mail Search's requirements from a previous section, along with recommended user interface objects, as well as the script suite to which each object class belongs (where you can examine the scripting terminology for that object):

- Mail Search can obtain and display a list of all mailboxes from all accounts.

  To display a list of mailboxes, consider using an outline view (from the Data View suite). An outline view can display hierarchical data, similar to the way the Finder displays folders and files in a list view. You can build the Outline sample application to see an outline view in action.

  To avoid unnecessary complexity in this tutorial, Mail Search only uses one layer of nesting in the outline view.

- A user can specify which mailboxes to search, including a single mailbox, a subset, or all mailboxes.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

An outline view, as a subclass of the table view class (both belong to the Data View suite), automatically supports row selection.

- A user can specify text to search for.

  A text field (from the Control View suite) is the obvious choice for obtaining text from a user.

  In addition, Mail Search can use a button (also from the Control View suite) to initiate the search.

- A user can specify where to search:

  - Subject field
  - To field
  - From field
  - Contents of message

  A good way to choose one item from a small number of items is a pop-up menu, which is supported by the popup button class (from the Control View suite). By using a pop-up menu, Mail Search mimics Mail's implementation of this feature.

- Mail Search can find and display all messages containing the specified text.

  There are several aspects of a found message that a user might like to see, including who it's from, the subject, and the mailbox where it is stored. In fact, the interface for displaying messages should look similar to the ones that display mail in Mail and similar applications.

  A table view (from the Data View suite) is a good choice for displaying rows and columns of data. You can build the Tables sample application (distributed with AppleScript Studio) to see a table view in action. The Tables application demonstrates two ways to work with table views: using a data source object to supply table data (the approach Mail Search uses), and operating without a data source (supplying the data directly from a script).

  A data source object is a special object supplied by AppleScript Studio to supply row and column data to a table or outline view. Data source objects are available from the AppleScript pane of Interface Builder's Palette window.

- A user can double-click a message in the result list to open it in a separate window.

  As a simple solution for displaying text in a separate window, you can use a window object containing a text view (from the Text View suite). Though it is easy to implement, a text view supports many operations on the displayed messages, including Cut, Paste, and even Undo.

- During long operations, such as searching through multiple mailboxes and potentially thousands of messages, Mail Search should provide feedback on its progress.

  A progress bar (from the Control View suite) provides a standard mechanism for user feedback. Combined with a text field, it can provide both determinate (the total time is known and the bar moves from left to right proportional to the percentage of the task completed) and indeterminate (the total time is not known, and a striped cylinder spins continually) progress bars, as well as text messages.

  > **Note:** The AppleScript Studio terminology for a progress bar is `progress indicator`; that term is used in Mail Search's script file.

You've now got a good start on the objects you'll use for Mail Search's user interface. In later sections, you'll read about a few additional objects you'll need that work together with those described here.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

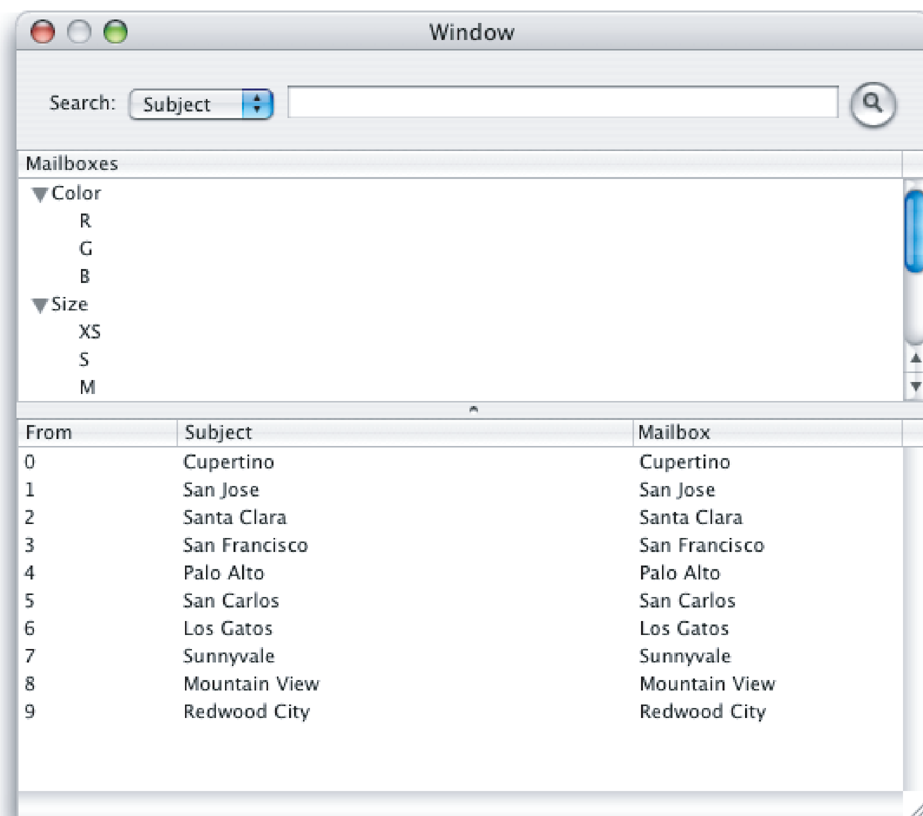
Arrange the User Interface

There are any number of ways to sketch out a user interface, from the legendary napkin sketch to laying out actual interface objects in a tool such as Interface Builder. This section presents Interface Builder snapshots of the prospective Mail Search interface. Keep in mind that these illustrations represent just one solution for the specified requirements. Other solutions are certainly possible; feel free to look for changes and improvements as you work through this tutorial and gain knowledge of the interface objects available in AppleScript Studio.

Figure 6-2 shows the Interface Builder definition for Mail Search's main window, the search window. It contains a pop-up menu to specify where to search, a text field to specify the text to search for, a button to start the search, an outline view to show the available mailboxes, and a table view to display the messages found by the search. The outline view and table view contain sample entries inserted by Interface Builder to help display the views' rows and columns. This data is not displayed in the application.

You'll learn how to create each of the interface objects shown in the search window in "Create the Search Window" (page 152). The search window also contains the standard close, minimize, and zoom buttons. This set of buttons is just one of the many features you'll get automatically in an AppleScript Studio application.

Figure 6-2 Mail Search's search window in Interface Builder



2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Figure 6-3 shows the status dialog that Mail Search displays during lengthy operations, such as gathering a list of mailboxes or searching through a large number of messages. Mail Search displays the status dialog as a sheet attached to the main window, not as a separate window. You'll learn how to create each of the interface objects shown in the status dialog in "Create the Search Window" (page 152).

Note: You'll have to get use to a bit of naming inconsistency when working with the status dialog. The Mail Search application often refers to the status dialog as a status panel because you use an object that Interface Builder calls a Panel to create the status dialog. Whether you see status dialog or status panel, you'll know it refers to the same object.

Figure 6-3 A status dialog in Interface Builder

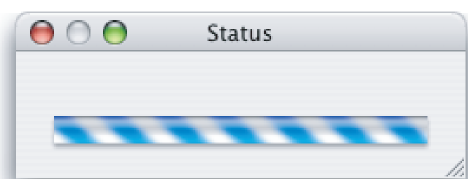


Figure 6-4 shows the MainMenu instance from Mail Search's `MainMenu.nib` file as it appears in Interface Builder. The application menu is open, showing the items in that menu. Mail Search uses the default menu nib created for an AppleScript Studio application, changing only the names of certain menu items shown in Figure 6-4. You'll learn how to create this nib in "Create the Message Window" (page 137) and how to modify it in "Customize Menus" (page 209).

Figure 6-4 Mail Search's menu nib in Interface Builder, showing the application menu

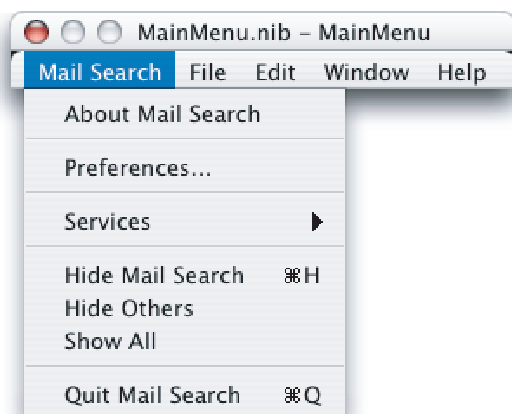
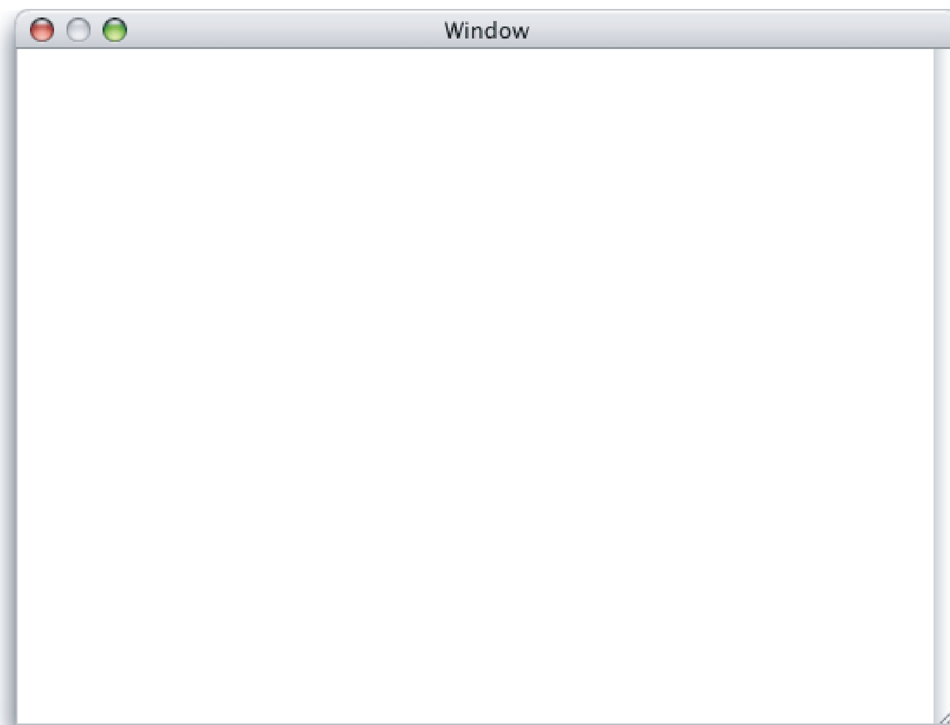


Figure 6-5 shows a Mail Search message window as it appears in Interface Builder. When a user double-clicks a message in Mail Search's search window, Mail Search opens a message window to display the message text. You will learn how to create this window in "Create the Message Window" (page 137).

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Figure 6-5 A Mail Search message window in Interface Builder



The interface items shown in this section form the basic user interface for the Mail Search application.

Plan the Code

Although you won't write scripts and handlers for Mail Search until you've completed several additional tasks, it makes sense to spend a little time now thinking about the application's code. A look at dependencies between the code and the interface provides useful background for building the interface and connecting it to handlers in application script files.

> **Note:** Mail Search has a tiny dab of Objective-C code in its `main` routine, and there is a lot of Cocoa code working behind the scenes to implement user interface objects and application operations. But the code you'll design for Mail Search consists of only handlers and script objects, and the script statements they contain.

Because every AppleScript Studio application is built on the Cocoa application architecture, Mail Search can perform many operations automatically, without any additional effort on your part. For example, users can open multiple windows, resize and minimize windows, enter text, and even shuffle the position of the From, Subject, and Mailbox columns in the list of found messages. To experiment with the features you get in the simplest document-based AppleScript Studio application, even before adding any of Mail Search's user interface, follow the steps in "Create a Project" (page 133).

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

AppleScript Studio applications also have the ability to connect user actions and other events in the application to handlers in scripts. As you saw in "Terminology Browser" (page 51), user interface objects in AppleScript Studio applications can respond to a variety of events and contain many scriptable elements and properties. In planning the code for Mail Search, you'll need to identify the events that Mail Search responds to.

To summarize Mail Search briefly, it opens a search window, connects to the Mail application (opening it if necessary), obtains a list of available mailboxes, and displays them. At that point, it waits for user input and responds accordingly, by opening new search windows, selecting mailboxes to search, initiating a search, displaying results, and so on.

To perform these operations, Mail Search needs two kinds of handlers:

- Handlers that respond directly to user actions (such as the opening or closing of a window) or changes of state in the application (such as completion of application launch). These handlers, called event handlers (as defined in "Connecting Actions to Scripts" (page 22)), are identified in "Event Handlers in Mail Search" (page 128).
- Handlers that are not necessarily called to respond to an event, but rather perform basic tasks in the application, such as obtaining a list of available mailboxes, displaying a status dialog, and so on. These handlers are typically called from event handlers. These handlers are identified in "Additional Handlers and Scripts in Mail Search" (page 129).

Because AppleScript Studio provides so much built-in support, Mail Search won't require as much code as you might expect. For a full listing of Mail Search's script file, `Mail Search.applescript`, see "Mail Search Tutorial: Write the Code" (page 187).

Important: Before implementing any of the handlers or other script statements shown in the Mail Search tutorial chapters, you should read the section "Obtaining the Code for the Mail Search Tutorial" (page 187).

Event Handlers in Mail Search

Most interaction with user interface items takes place in the search window, and you've already identified the objects for that window in "Identify Objects for the User Interface" (page 123). Working from those objects, you can identify events the objects must handle and the event handlers for those events:

- **The search window:** Mail Search needs to know when a window is opened, when it is activated, and when it is closed. The corresponding event handlers are:
  - `will open`: This event handler is called after a window is instantiated from a nib file and before it is opened. Mail Search can use this handler to perform any initialization associated with the search window that can't be stored with the nib file.
  - `became main`: This event handler is called when a window becomes the main window (the front window and principal focus of user action), such as when a window is first opened or when a user switches back to the application. The main window is typically (but not always) also the key window (or first recipient for keystrokes). Mail Search can use this handler to perform any required tasks prior to the search window becoming main, such as checking whether the available mailboxes have been loaded.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

- ❏ `will close`: This event handler is called before a window is closed. Mail Search can use it to perform any cleanup when a search window is closed.

- **The search text field:** Mail Search needs to know when a user presses the Return key so it can initiate a search. It can do so by implementing an `action` event handler.
- **The find button:** Mail Search also needs to know when a user clicks the find button so it can initiate a search. It can do so by implementing a `clicked` event handler.
- **The search results table view:** Mail Search should open a message window when a user double-clicks on a message. It can handle this event by implementing a `double-clicked` event handler.
- **The application object:** Mail Search needs to perform certain initialization tasks after the application has unarchived its user interface objects from the nib file but before it enters its main routine. It can achieve this goal by implementing a `will finish launching` event handler.

You'll connect objects to these event handlers in "Connect the Interface" (page 169) and write the handlers in "Write Event Handlers for the Interface" (page 188).

Additional Handlers and Scripts in Mail Search

A **script object** is a user-defined object, combining data (in the form of properties) and handlers, that can be used in a script. A **script object definition** is a compound statement that can contain collections of properties, handlers, and other AppleScript script statements. A script object definition is similar to an object-oriented class definition—you can instantiate multiple instances of the script object, each containing data and handlers to operate on that data. You can even extend or modify the behavior of a handler in one script object when calling it from another script object.

Most event handlers in Mail Search are associated with the search window. These event handlers need to call other handlers to perform searches and display results. One convenient way to organize these handlers is to create a script object for each search window that implements all the necessary handlers related to searching and displaying results. In response to user actions, Mail Search's event handlers call the script object's handlers, which in turn call other handlers in the script object as needed. Mail Search can also use a script object to encapsulate operations involving the status dialog. In Mail Search, neither of these script objects require the use of inheritance to modify contained handlers.

In the next sections, you'll specify script and handlers for Mail Search. You'll write these handlers in "Write Scripts and Additional Handlers" (page 192).

The Controller Script

Mail Search defines a script to handle operations for the search window. It calls this script a controller, in the tradition of the **model-view-controller (MVC)** paradigm. In MVC, the view is responsible for what the user sees, the model represents the application's data and algorithms, and the controller interprets user input and specifies changes to the model and the view. When a search window is about to open, Mail Search creates a controller script object for it and stores it in a global list of controllers. When a window is activated, Mail Search gets its controller and tells it to load all available mail- boxes from the Mail application.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Each open search window has an associated controller in Mail Search's global list of controllers. When a user performs an action in a window, such as clicking the find button, the application calls the appropriate event handler (such as `on clicked`). That event handler typically gets the controller (a script object) for its window, then calls the appropriate handler in the controller (such as the `find` handler) to perform the requested action (to find matching messages).

The controller script object defines properties for things it needs to keep track of, including

- a reference to its window
- a reference to a status panel (or dialog)
- a list of found messages
- a boolean for whether it has created a list of available mailboxes

The controller script object defines handlers for the searching-related tasks it performs:

- `initialize`: performs any initialization needed for searching
- `loadMailboxes`: loads mailboxes if they haven't already been loaded; shows status dialog
- `find`: if a valid search has been specified (there is text to search for and at least one selected mailbox to search in), performs the search

In addition to these high-level handlers, the controller script object needs handlers to actually load the mailboxes and search for and display messages. For example, to load mailboxes, Mail Search must search each account. This leads to the following handlers:

- `addMailboxes`: called by `loadMailboxes`; calls `addAccount` for each account
- `addAccount`: called by `addMailboxes`; adds an account name to the mailboxes view; for each mailbox in the account, calls `addMailbox`
- `addMailbox`: called by `addAccount`; adds the mailbox to the mailboxes view

This should help you understand how the controller script might implement other tasks, such as adding found messages to the messages view.

Finally, Mail Search needs several handlers that are not part of the controller script itself:

- `makeController`: creates a controller script object for a window
- `addController`: adds a controller to the global list of controllers
- `removeController`: removes a controller from the global list of controllers
- `controllerForWindow`: given a window, returns the controller for that window from the global list of controllers

You'll find more information about controller handlers in "Write Scripts and Additional Handlers" (page 192). Of course you can always look ahead by examining Mail Search in "Mail Search Tutorial: Write the Code" (page 187), or by opening the Mail Search sample application in Xcode.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

The Status Dialog Script

The status dialog provides both determinate (the bar moves from left to right) and indeterminate (a spinning striped cylinder) progress bars, as well as text messages. Mail Search's approach for handling a status dialog is similar to its approach for handling search operations. That is, it defines a status dialog script and instantiates a script object based on that script whenever it needs to display a status dialog. The status dialog script defines properties for things it needs to keep track of, including

- a reference to its window
- a boolean for whether it has been initialized
- count variables used in showing progress

The status dialog script is fairly simple, and requires fewer handlers than the controller script. As you might expect, its handlers are used for opening and closing the panel or adjusting its status:

- `openPanel`: performs initialization and displays the status dialog
- `closePanel`: closes the panel
- `changePanel`: changes the progress display message
- `adjustPanel`: adjusts the current state of the progress bar
- `incrementPanel`: increments the progress bar display

These handlers aren't shown in individual listings, but you can examine them in "Mail Search Tutorial: Write the Code" (page 187) or in the Mail Search sample application.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Mail Search Tutorial: Create the Interface

Mail Search is a fairly complex AppleScript Studio application that searches for specified text in messages in the Mac OS X Mail application. In this chapter you'll create an AppleScript Studio project for Mail Search and build the user interface. These tasks are described in the following sections:

1. "Create a Project" (page 133).
2. "Add an Image File to the Project" (page 135)
3. "Build the Interface" (page 136).

This chapter assumes you have completed "Mail Search Tutorial: Design the Application" (page 119) the previous Mail Search tutorial chapter.

Create a Project

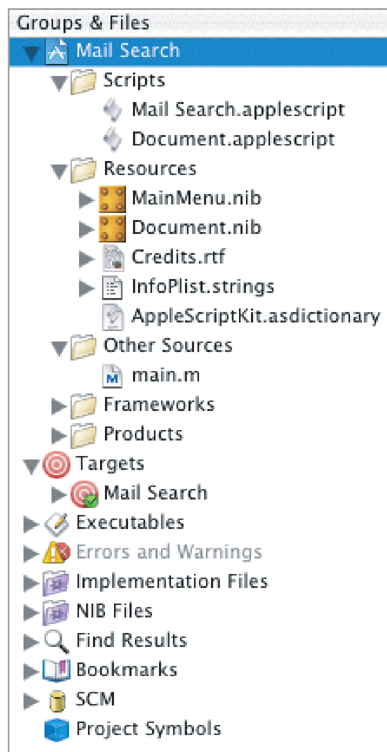
To create a new AppleScript Studio project for Mail Search, perform these steps:

1. Use Xcode to create a new AppleScript Studio project, as described in "Creating a Hello World Application" (page 27), but choose the AppleScript Document-based Application for the project template—you use this template for an application that creates and manages multiple documents—and name the project "Mail Search".

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

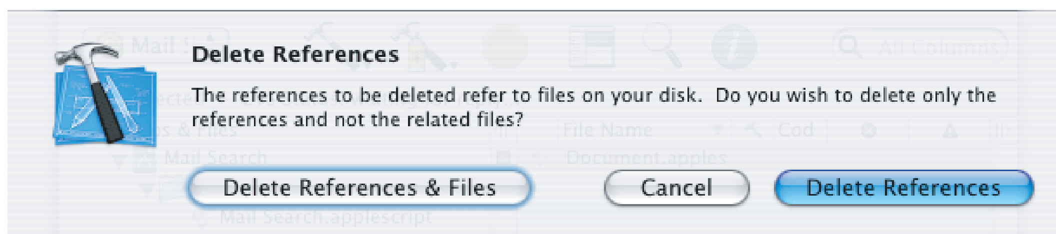
2. Open the groups in the Files list in the Groups & Files list to show their contents, as in Figure 7-1. Each of the items is described in "Default Project Contents" (page 44).

Figure 7-1 Default contents of a document-based AppleScript Studio project



3. `Document.applescript` is the default document script file, which is created as an empty file. Mail Search doesn't use this file, so select the file and press the Delete key to remove it from the project. You'll then see the dialog shown in Figure 7-2.

Figure 7-2 Deleting a file from a project



Click the Delete References & Files button to delete the file on disk, as well as from the project.

4. Select the script file `Application.applescript` and choose Rename from the Project menu. Rename the file `Mail Search.applescript`.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

At this point you can build the project and create a working application that can create multiple document windows, expand and minimize them, display an About window, and respond to a number of menu choices. To build and run the application, do one of the following:

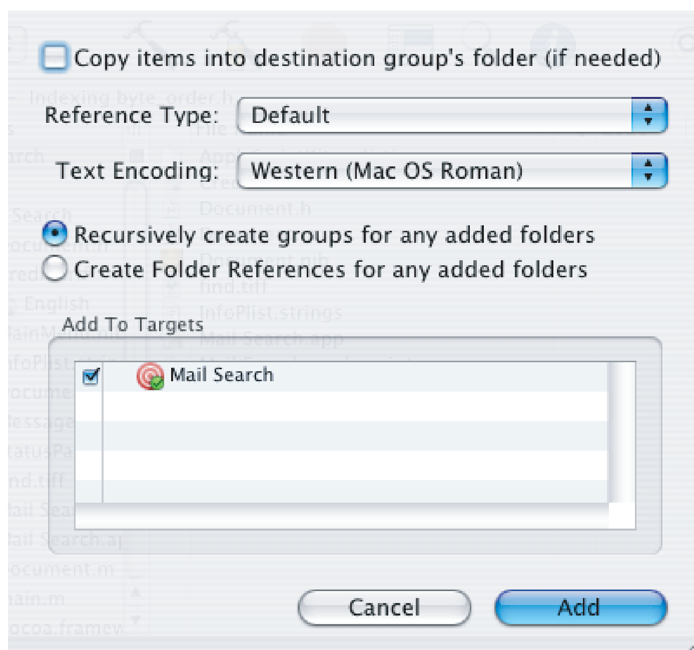
- type Command-R
- choose Build and Run from the Build menu
- click the Build and Run button, which is represented by a hammer and green go button

Add an Image File to the Project

The Mail Search sample project included with AppleScript Studio includes an image of a magnifying glass you'll need for the search window. Open the folder for the Mail Search sample project and drag the file `find.tiff` to the Files list in the Groups & Files list of your new Mail Search project. You can drag it directly to the Resources group.

You can also add a file to the Mail Search project by choosing Add Files from the Project menu in Xcode, then navigating to the file `find.tiff` in the directory for the Mail Search sample project. Select it and click Open. Whichever approach you choose, you'll get the dialog shown in Figure 7-3.

Figure 7-3 Adding a file to a project



Select the checkbox "Copy items into destination group's folder (if needed)," then click Add to add the file to your project. If you used the Add Files menu choice, drag the file `find.tiff` into the Resources group in the Groups & Files list.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Build the Interface

You'll build the interface for Mail Search with Interface Builder, a Mac OS X development tool located in `/Developer/Applications`. You should be familiar with Interface Builder from reading about it in "Interface Builder Features for AppleScript Studio" (page 53), and from completing the "Currency Converter Tutorial" (page 89). In particular, you should be familiar with the nib files Interface Builder uses to store interface definitions.

Each of the following sections provides instructions for building one of the interface items described in "Design the Interface" (page 123). You'll start with the simpler items and with nib files that are created automatically as part of the project, then move to more complex interface items that require the creation of new nib files.

As you work through the steps to create Mail Search's interface, you can build and run the application at any point. You won't see much difference in the application's interface until you've completed the section "Create the Search Window" (page 152). That's because except for the menus and the main search window, the interface won't become visible until after you connect it to event handlers and write the handlers in later sections.

To Build the interface for Mail Search, you'll work through these steps:

1. "Examine the Default Menus" (page 136)
2. "Create the Message Window" (page 137)
3. "Create a Status Dialog" (page 145)
4. "Create the Search Window" (page 152)

Examine the Default Menus

You won't make any changes to Mail Search's menus until "Customize Menus" (page 209). In this section you'll examine the menus in Interface Builder. To examine the default menus in a Document-based AppleScript Studio project template, you do the following:

1. Open the Mail Search project in Xcode.
2. Open the `MainMenu.nib` file by double-clicking its icon in the Files list in Xcode's Groups & Files list, shown in Figure 7-1 (page 134).

   Interface Builder opens, displaying the four windows shown and described in "Interface Creation" (page 54).

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Figure 7-4 Interface Builder windows after opening Mail Search's `MainMenu.nib` file

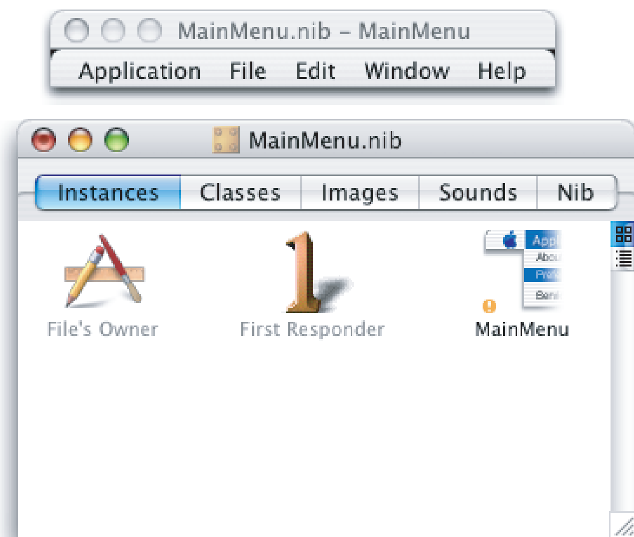


Figure 7-4 shows just the MainMenu.nib window and the application's menus. The four icons in the Instances pane visible in Figure 7-4 are described in "Interface Creation" (page 54). The File's Owner instance represents NSApp, a global constant for the application object that serves as the master controller for the application. You'll use this instance in "Connect the Application Object" (page 169).

At this point, you can use any of Interface Builder's capabilities for creating and modifying menus, such as adding or deleting menus or menu items. You don't need to do anything now to change the default menus. But in "Customize Menus" (page 209), you'll find steps for changing menu and menu item names so that they match the menus shown in Figure 6-4 (page 126). When you are finished looking at the menus, close the nib file window; otherwise, you'll end up with a lot of open nib windows as you work through the tutorial.

Create the Message Window

Figure 6-5 (page 127) shows the design for a Mail Search message window as it appears in Interface Builder. The message window is a simple window with one main view and no special features. To create the message window, you need to:

1. Create a new nib file.

2. Add a window instance to the nib file.

3. Add interface objects (in this case, a single text view object, enclosed within a scroll view) to the window.

These steps are described in the following sections.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Create a Nib File

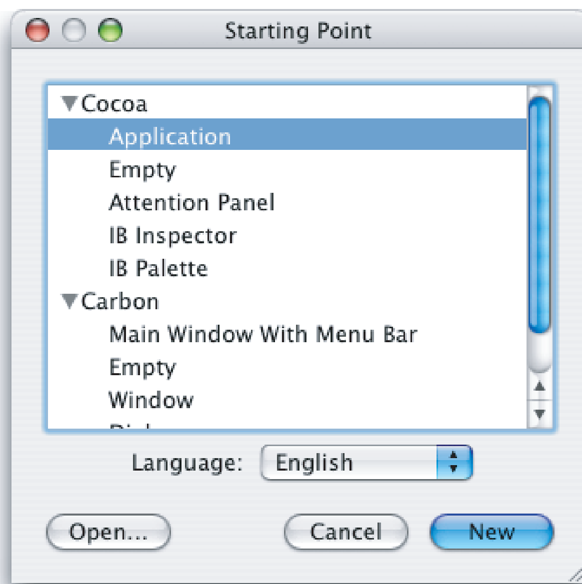
To create a new nib file, perform these steps:

1. Open the Mail Search project in Xcode.

2. Double-click one of the nib file icons (such as `MainMenu.nib`) in the Files list in the Groups & Files list to start Interface Builder.

3. In Interface Builder, choose New from the File menu. Interface Builder opens the dialog shown in Figure 7-5.

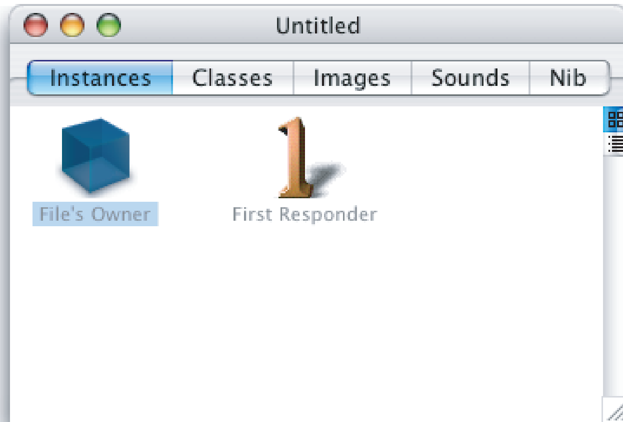
Figure 7-5 Creating a new nib file in Interface Builder



2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

4. Select Empty in the Cocoa section and click New. The new nib file is shown in Figure 7-6. The icons in the Instances pane visible in Figure 7-6 are described in "Interface Creation" (page 54).

Figure 7-6 A new nib file in Interface Builder



5. Choose Save As from the File menu and navigate to the `English.lproj` directory in your Mail Search project. Save the file as `Message.nib` (you just type "Message" and Interface Builder adds the extension). You will be prompted for a nib file format. Your three choices are:

   - Pre-10.2 format: Use this if you want your application to run in Mac OS X versions earlier than 10.2.
   - 10.2 and later format: Use this if you want your application to run in Mac OS X version 10.2 and later.
   - Both formats: Use this if you want to run your application in all versions of Mac OS X.

   Select the option for 10.2 and later.

6. After saving the file, you should be prompted to add the file directly to the Mail Search project. Click Add to add the nib file to the project.

   If for any reason you don't have the opportunity to automatically add the new nib file to the Mail Search project, you can add it directly by opening the project in Xcode, and dragging it into the Resources group in the Groups & Files list, which is shown in Figure 7-1 (page 134), or by choosing Add Files from the Project menu, navigating to the `Message.nib` file, and choosing that file. If you use Add Files, you should then drag the icon for the nib file to the Resources group.

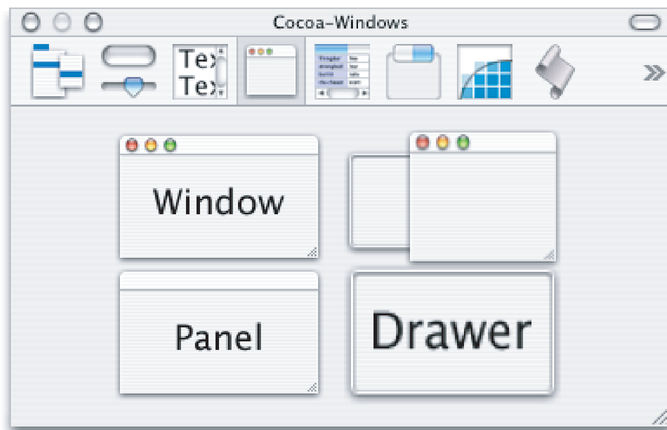
Add the Message Window to the Nib File

To add a message window to the nib file, perform these steps:

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

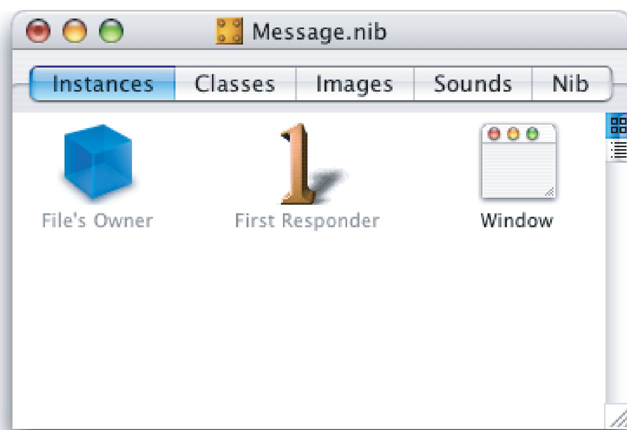
1. In Interface Builder, click the Cocoa Windows button in the Palette window toolbar. The Palette window, showing the Cocoa Windows palette, is shown in Figure 7-7.

Figure 7-7 The Cocoa-Windows palette of Interface Builder's Palette window



2. Drag an instance of Window from the Palette window to any convenient space on your desktop. You'll see a window that looks similar to the image you dragged from the Palette window. You'll also see an icon for the window added to the Message.nib window, as shown in Figure 7-8.

Figure 7-8 The Message.nib window, showing a window instance



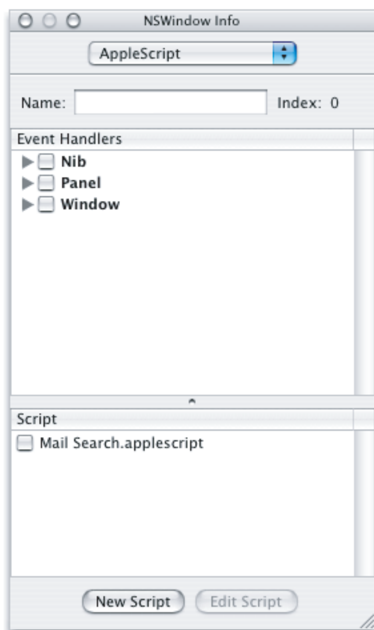
3. Double-click the word "Window" in the window instance in the nib window and type "Message" to change the instance name.

4. You need to provide an AppleScript name for the window too, so you can access it in scripts. To do so, click in the nib window to select the Message window instance, then choose Show Info from the Tools menu or type Command-Shift-I to open the Info window. Use the pop-up menu at the top of the window to display the AppleScript pane. The result is shown in Figure 7-9.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Note: You can use command shortcuts to choose the AppleScript pane or other panes, but the exact shortcut can vary between releases of Interface Builder. For example, in Interface Builder version 2.4.2, you can display the AppleScript pane by typing Command-8.

Figure 7-9 The AppleScript pane in the Info window for a window object



To name the window, simply type "message" in the Name field. Remember that this is the object's AppleScript name, which you can use in a script to identify the object. It is a different entity than the name of the window object in the nib window, and is also different from the window's title in the running application.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

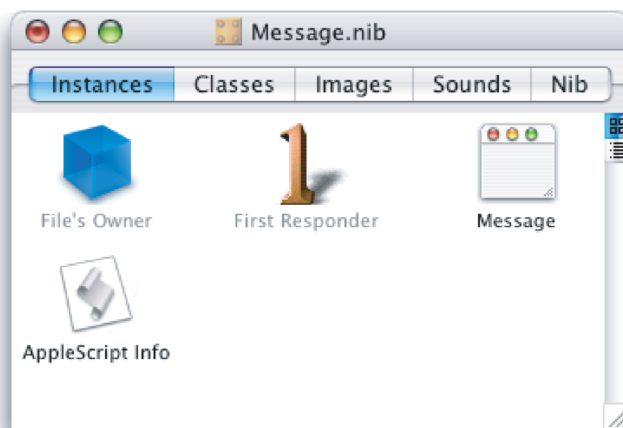
Note: The first time you make a change in the AppleScript pane, Interface Builder adds a new instance named AppleScript Info to the nib window. The type of this instance is ASKNibObjectInfoManager. An example is visible in Figure 7-10.

The AppleScript Info object stores information the application needs at runtime, such as object names and event handler connections. This information is saved with the nib file and retrieved when the application's objects are unarchived from the nib file at application launch.

You can delete the AppleScript Info instance if you want to remove all such information from your application. If you do so, you will then have to reconnect your event handlers.

If it makes sense, you can use the same AppleScript name for objects of different types in the same window (such as a button and a text field), or for objects of the same type (such as buttons) in different windows. However, if you name two buttons in the same window "button," you won't be able to differentiate between them by name in an application script.

Figure 7-10 The Message.nib window showing an AppleScript Info object (not selected)



5. Save the changes to the `Message.nib` file. You should save changes periodically as you work through this tutorial.

   When you are finished with a nib file, you should also close its window to avoid a clutter of nib-related windows as you work through the tutorial.

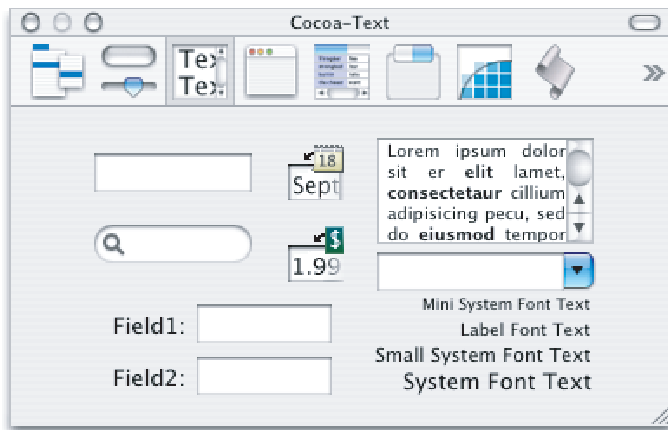
Set Up Interface Objects in the Message Window

To set up interface objects in the message window (in this case, there is just one object, a text view), perform these steps:

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

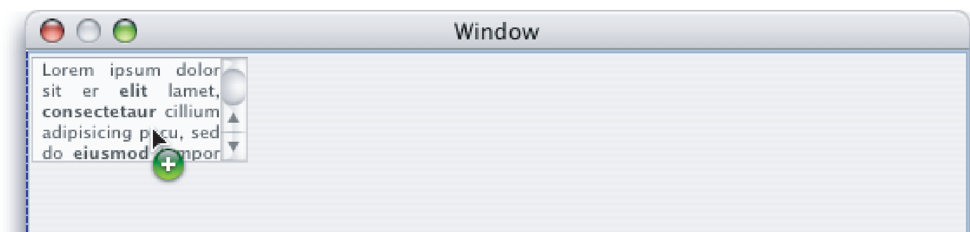
1. Click the Cocoa Text Views button in the Palette window toolbar. The Cocoa-Text palette is shown in Figure 7-11.

Figure 7-11 The Cocoa-Text palette of Interface Builder's Palette window



2. Drag a text view object (the item containing Latin text and one scroll bar) from the Palette window to the message window you created previously. Drag it to the top-left corner of the window (below the title bar). Interface Builder provides feedback to aid in alignment and resizing, as shown in Figure 7-12. The text view is actually enclosed within a scroll view with one scroll bar.

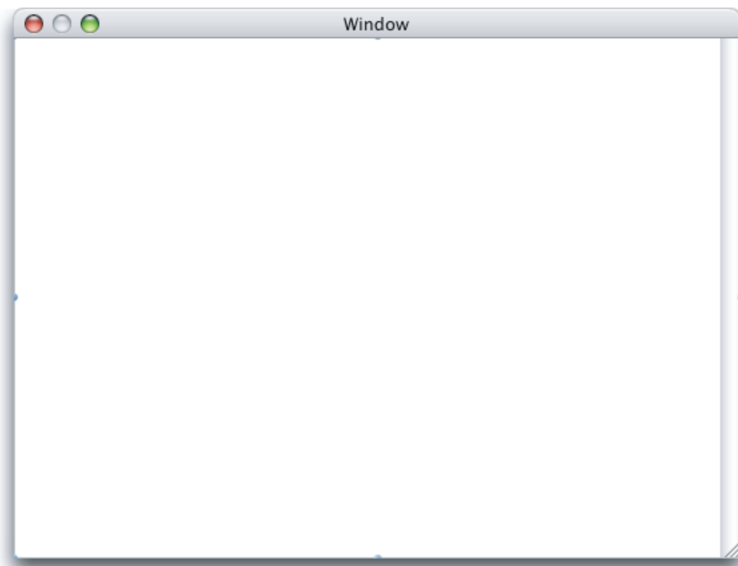
Figure 7-12 Positioning a text view object



2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

3. Drag the bottom-right corner of the text view object to resize it until it fills the whole window. Again, you get resizing feedback from Interface Builder. The result is shown in Figure 7-13. The small dots below the title bar and just inside the other sides of the window indicate that the text view is still selected.

Figure 7-13 The finished message window



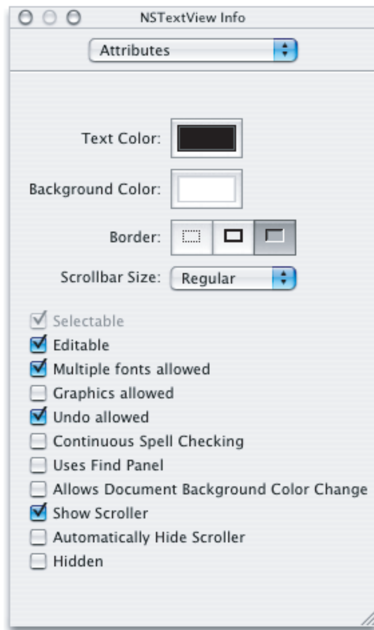
4. To provide an AppleScript name for the text view, open the Info window and display the AppleScript pane. The result is similar to Figure 7-9 (page 141). Type "message" in the Name field.

5. To support Undo in the text view, you'll have to modify its default attributes. While the Info window is still open, use the pop-up menu at the top of the window (or type Command-1) to display the Attributes pane.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

6. In the Options section, select the "Undo allowed" checkbox. The result is shown in Figure 7-14.

Figure 7-14 Attributes pane in Info window for text vie



7. As always, save the nib file after completing your changes.

Create a Status Dialog

Figure 7-15 shows a status dialog as it appears in Interface Builder. Mail Search displays the status dialog as a sheet on the search window during lengthy operations. The status dialog contains a progress bar and an invisible text field in which Mail Search can write status messages.

Note: As mentioned in "Arrange the User Interface" (page 125) the Mail Search application often refers to the status dialog as a status panel because you use an object that Interface Builder calls a Panel to create the status dialog. Whether you see status dialog or status panel, you'll know it refers to the same object.

To create the status dialog, you'll perform steps very similar to those in "Create the Message Window" (page 137). You need to:

1. Create a new nib file. This process is described in "Create a Nib File" (page 138).

   In this case, name the nib file `StatusPanel.nib`.

2. Use the same process described in "Add the Message Window to the Nib File" (page 139) to add a window instance to the nib, but with these differences:

   a. drag an instance of the user interface object labeled "Panel" from the Cocoa-Windows palette, rather than the one labeled "Window"

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

b. name the instance “Status”

c. type “status” in the Name field in the AppleScript pane of the Info window for the instance

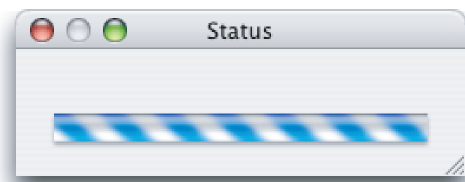
3. Adjust its size and attributes for the status dialog.
4. Set up a progress bar in the window.
5. Set up a text field to display progress text.

Steps 3, 4, and 5 are described in the following sections.

Adjust the Size and Attributes of the Status Dialog

You need to change certain window attributes to use the window as a pane. You also need to reduce the size of the status dialog so that it looks appropriate when it appears below the title bar of the search window. The previously designed status dialog, shown again in Figure 7-15, is smaller than the minimum size for a default window in Interface Builder, so you have to adjust the minimum size.

Figure 7-15 The status dialog as previously designed



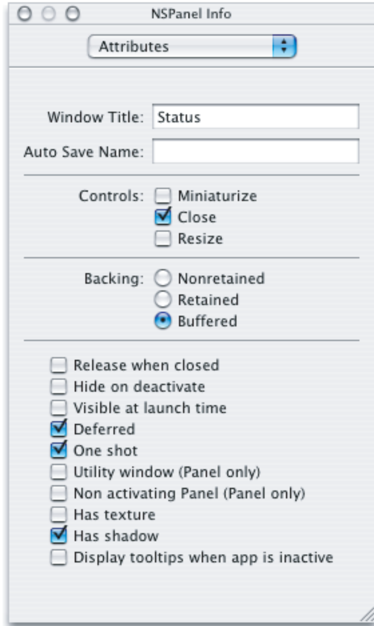
To make these changes, perform the following steps:

1. Select the Status window instance in the StatusPanel.nib window, then display the Attributes pane in the Info window.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

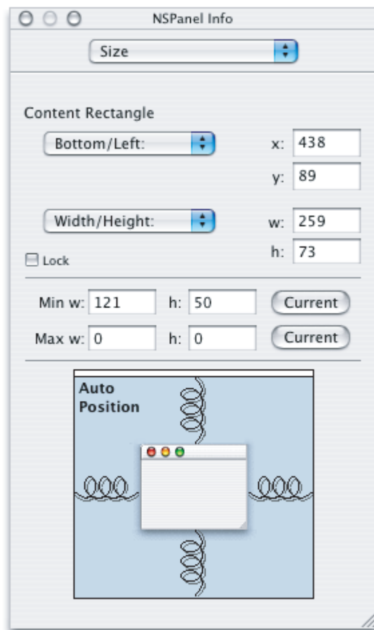
2. In the Window Title field, change the title from Panel to Status. The result is shown in Figure 7-16.

Figure 7-16 The revised Attributes pane in the Info window for the status dialog



3. With the Info window still open, use the pop-up menu at the top of the window to display the Size pane. The result is shown in Figure 7-17.

Figure 7-17 The Size pane in the Info window for the status dialog

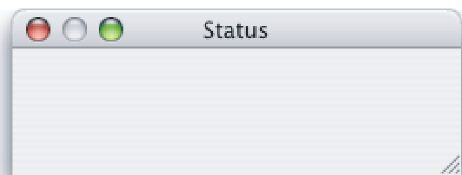


2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

4. In the Min Size section, type 110 for the width and 50 for the height. (These values just have to be at least as small as the values you'll enter in the next step.)

5. In the Content Rect section, type 260 for the width and 75 for the height. The resized status dialog that results from these changes is shown in Figure 7-18.

Figure 7-18 The resized status dialog

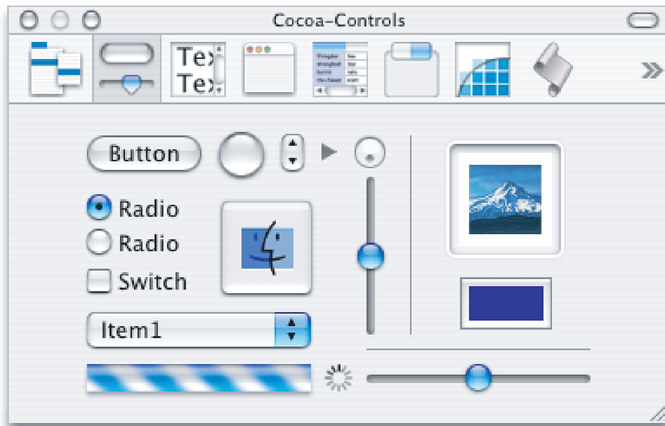


Set Up a Progress Bar

To set up a progress bar in the status dialog, do the following:

1. Click the Controls button in the Palette window toolbar. The Cocoa-Controls palette is shown in Figure 7-19.

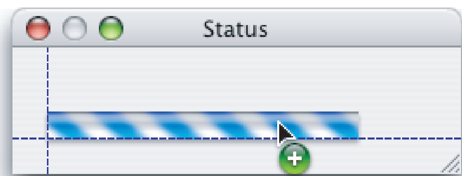
Figure 7-19 The Cocoa-Controls palette of Interface Builder's Palette window



2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

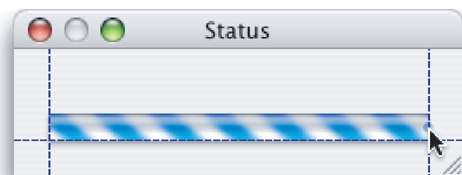
2. Drag a progress bar object (the horizontal striped cylinder near the bottom of the window) from the Palette window to the status dialog you just created and position it as shown in Figure 7-20.

Figure 7-20 Positioning the progress bar



3. Select the progress bar, then grab the lower-right corner and resize it. Use Interface Builder's feedback to help you resize the progress bar, as shown in Figure 7-21.

Figure 7-21 Resizing the progress bar



4. You need to provide an AppleScript name for the progress bar so you can access it in scripts. To do so, select the progress bar, then open the Info window to the AppleScript pane. The result is similar to Figure 7-9 (page 141).

> **Note:** As mentioned previously, the AppleScript Studio terminology for a progress bar is `progress indicator` and you'll see the class name "NSProgressIndicator" as part of the Info window title.

To name the progress bar, simply type "progress" in the Name field.

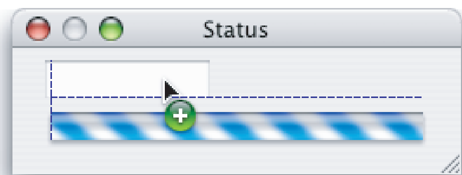
Set Up a Text Field

To set up a text field to display progress text in the status dialog, do the following:

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

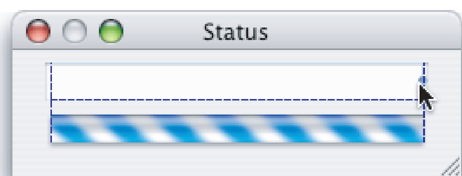
1. Drag a text field object (a rectangular, unlabeled field with a white background) from the Cocoa-Text palette (shown in Figure 5-10 (page 99)) in the Palette window to the status dialog. Position the text field above the progress bar you added in a previous step. Use Interface Builder's feedback to help you align the left side of the text field with the progress bar and just above it, as shown in Figure 7-22.

Figure 7-22 Positioning a status text field above the progress bar



2. Select the text field, then grab the middle selection handle on the right and resize the field, as shown in Figure 7-23.

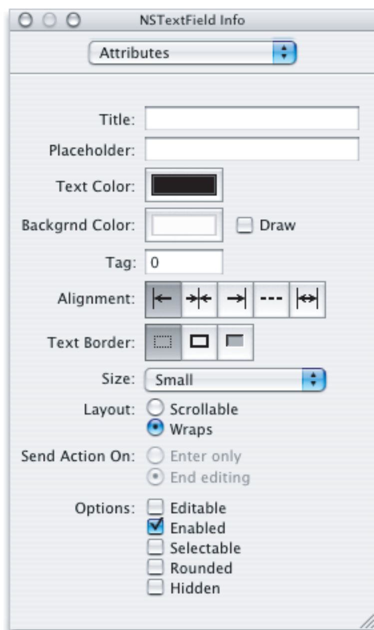
Figure 7-23 Resizing the status text field



2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

3. You don't want a user to enter text in the text field and you do want the field (though not the status messages that get written to it) to be invisible. To choose the correct settings, select the text field and open the Info window to the Attributes pane. The result is shown in Figure 7-24.

Figure 7-24 The Attributes pane in the Info window for the text field



Now make these adjustments, in the order shown:

a. Deselect the Editable checkbox.

b. Deselect the Selectable checkbox.

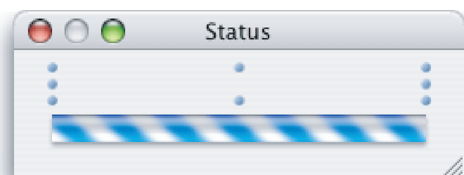
c. Select the Small option in the Size pop-up menu.

d. In the Border section, click the button on the left to select no border.

e. In the Backgrnd Color section, deselect the Draw checkbox.

f. In the Layout section, select the Wraps radio button. The resulting invisible text field is shown in Figure 7-25.

Figure 7-25 The invisible status text field



2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

4. You need to provide an AppleScript name for the text field so you can access it in scripts. To do so, display the AppleScript pane in the Info window, then type “statusmessage” in the Name field.

5. Save all the changes to the `StatusPanel.nib` file.

Create the Search Window

Now that you’ve used Interface Builder to create and modify several relatively simple nibs, you’ll get a chance to work with a more interesting and challenging example, the nib for Mail Search’s search window. Figure 6-2 (page 125) shows the design for the search window. To put together a nib that implements that window requires the following steps:

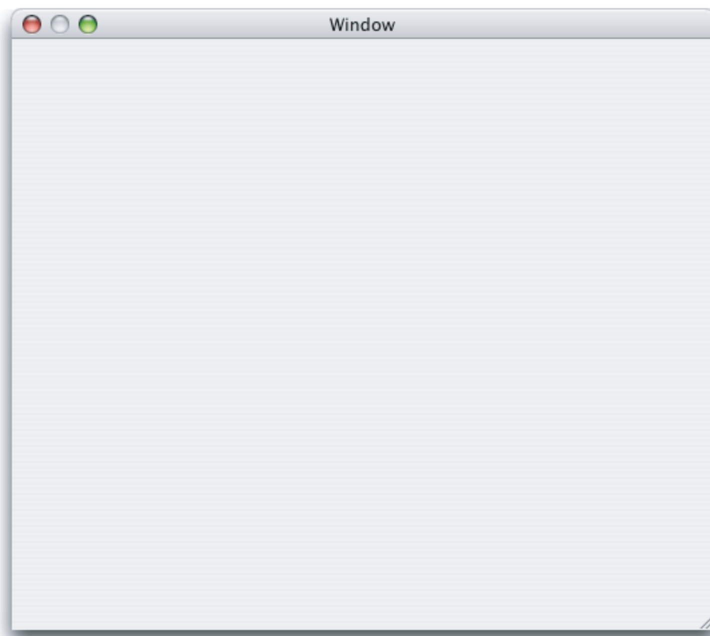
1. Open the nib file `Document.nib`.

   The steps for opening a nib file are listed in “Examine the Default Menus” (page 136).

2. Rename the default window instance in the nib window.

   Double-click the instance name “Window” and type “Mail Search” as the new name. The Mail Search search window is a little larger than the default window size (with a width of about 525 and a height of about 440). You can resize with the same steps you used in “Adjust the Size and Attributes of the Status Dialog” (page 146), though in this case you won’t need to reset the minimum size for the window. The resulting window is shown in Figure 7-26.

Figure 7-26 The resized, empty search window



2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

You should also follow the same steps you used in "Add the Message Window to the Nib File" (page 139) to display the AppleScript pane in the Info window, then type "mail search" in the Name field to supply an AppleScript name for the window.

3. Set up a popup button for the search type.

4. Set up a text field for the search text.

5. Set up a button to initiate searches.

6. Set up an outline view to display the available mailboxes.

7. Set up a table view to display matching messages.

8. Group the outline and table views in a split view.

Steps 3 through 8 are described in the following sections.

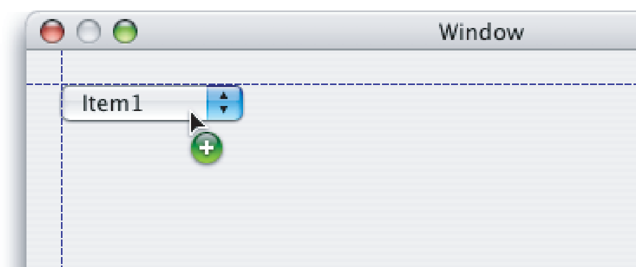
Once you've completed these steps, you can build the application, view the search window, and perform operations such as opening the pop-up menu, resizing the table and outline views, and rearranging the columns in the search result table. You can also build the application periodically to view the search window at various stages of completion. However, you won't be able to do any searching until you connect scripts to the user interface in later sections. To build and run the application, open the project in Xcode and type Command-R, choose Build and Run from the Build menu, or click the Build and Run button.

Set Up a Popup Button

Mail Search uses a popup button to provide a pop-up menu of search locations (in the To, From, or Subject fields of messages or in the message contents). To set up a popup button in the search window (the Mail Search window instance in the Document nib), perform these steps:

1. With the search window still open as shown in Figure 7-26, drag a popup button from the Cocoa-Controls palette in the Palette window (shown in Figure 7-19 (page 148)) to the search window. Use the automatic feedback to position the button in the top left corner of the window. As the button nears the corner, Interface Builder provides dashed feedback lines to help align the object according to the Aqua guidelines, as shown in Figure 7-27.

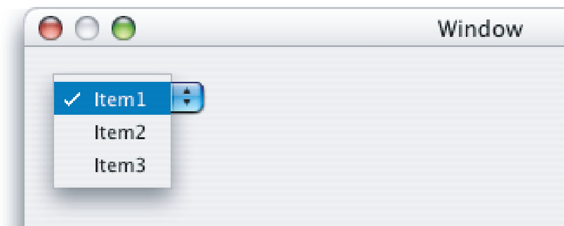
Figure 7-27 Positioning a popup button in the search window



2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

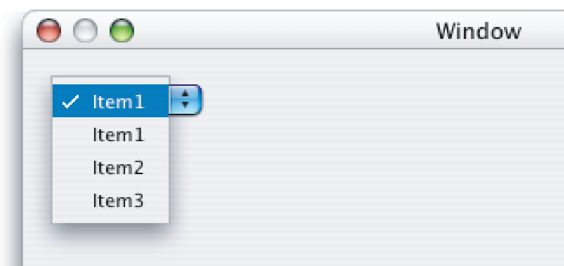
2. You need to provide menu items in the popup button for the possible search locations: Contents, Subject, To, and From. Double-click the button to reveal its default contents, as shown in Figure 7-28.

Figure 7-28 The default contents of a popup button



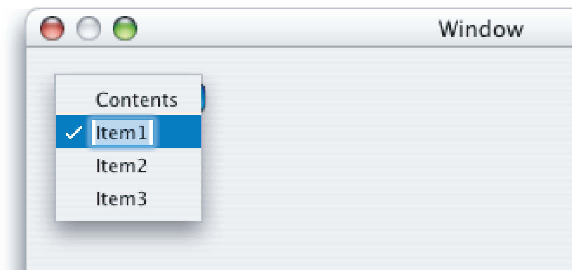
3. To add a fourth item, simply choose Copy from the Edit menu (copying the selected Item1) and then Paste from the Edit menu. The result is shown in Figure 7-29.

Figure 7-29 A popup button with a new item



4. To rename the first item, double-click it, type "Contents" as the new text, then tab to the next field. The result is shown in Figure 7-30.

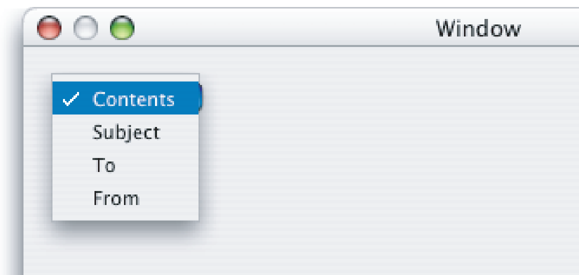
Figure 7-30 A renamed popup button item



2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

5. Rename the remaining three items as Subject, To, and From. Figure 7-31 shows the renamed items.

Figure 7-31 Popup button with renamed items

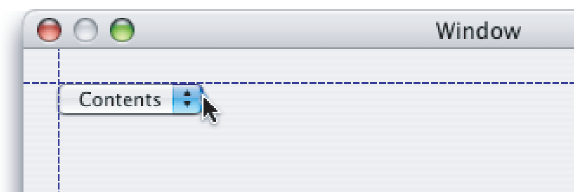


6. Open the Info window for the popup button to the Attributes pane and select the Small checkbox in the Options section. The result is shown in Figure 7-32.

Figure 7-32 The popup button after checking the Small checkbox

7. Adjust the size of the button so that it is just large enough for the item with the longest name, Contents. To do so, select the popup button, then drag the middle selection handle on the right side. As you resize, Interface Builder again provides alignment guides, as shown in Figure 7-33.

Figure 7-33 Resizing the popup button



8. You should also follow the same steps you used in "Set Up a Progress Bar" (page 148) to display the AppleScript pane in the Info window, then type "where" in the Name field to supply an AppleScript name for the popup button.

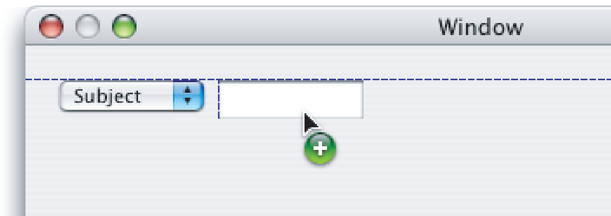
Set Up a Text Field

Mail Search uses a text field to allow a user to provide text to search for. To set up a text field in the search window, perform these steps:

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

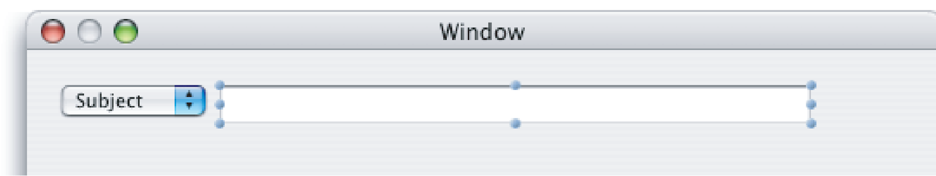
1. Drag a text field from the Cocoa-Text palette in the Palette window to the search window. Use the automatic feedback to position the field in the top left corner of the window next to the previously placed popup button, as shown in Figure 7-34.

Figure 7-34 Positioning a text field in the search window



2. To resize the text field, select it, then grab the middle selection handle on the right. Stretch the text field across the window, leaving room on the right for the find button. The result is shown in Figure 7-35

Figure 7-35 Resized text field in the search window



3. While the text field is still selected, open the Info window to the Attributes pane. In the Send Action On section, click the radio button for Enter only. With this setting, the Mail Search application begins searching if a user types search text and presses the Return or Enter keys, in addition to when a user clicks the find button. The keystrokes only initiate a search when the text field has the focus. (See "Setting the Keyboard Focus" (page 87) for related information.)

4. In the same window, select the Small option in the Size pop-up menu, as you did previously for the popup button.

5. While you still have the Info window open for the text field, display the AppleScript pane, then type "what" in the Name field to supply an AppleScript name for the text field.

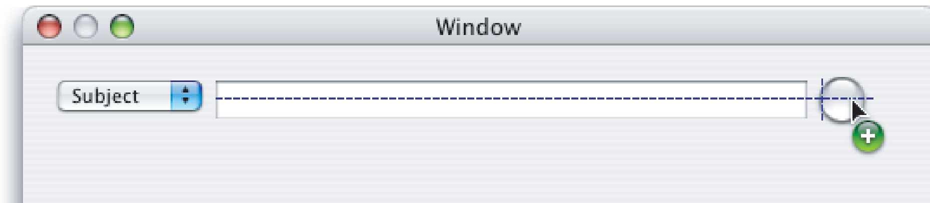
Set Up a Button

Mail Search uses a button to initiate a search. To set up a button in the search window, perform these steps:

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

1. Drag a round button from the Cocoa-Controls palette in the Palette window to the search window. Use the automatic feedback to position the button in the top right corner of the window next to the previously placed text field, as shown in Figure 7-36.

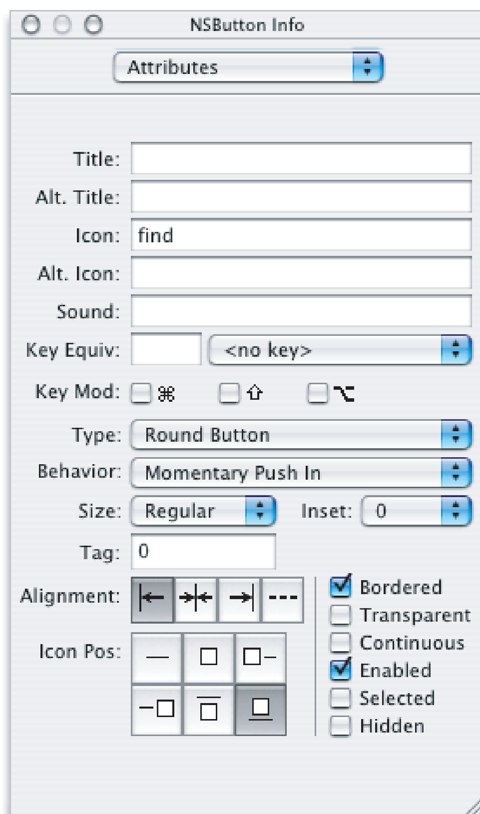
Figure 7-36 Positioning a button in the search window



2. Open the Info window for the button, display the AppleScript pane, then type "find" in the Name field to supply an AppleScript name for the button.

3. In "Create a Project" (page 133), you added an image file named `find.tiff` to the Mail Search project. That image file contains an image of a magnifying glass that you will now add to the button. With the Info window still open from the previous step, display the Attributes pane, then type "find" (the name of the image file, without the extension) into the "Icon:" field, as shown in Figure 7-37.

Figure 7-37 The Info window for the find button

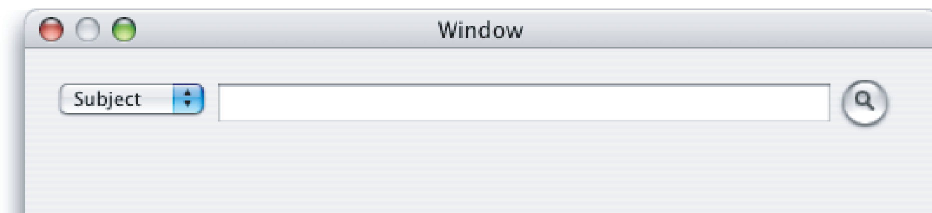


2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Alternatively, you can simply drag the image from the Images pane in the nib window to the find button.

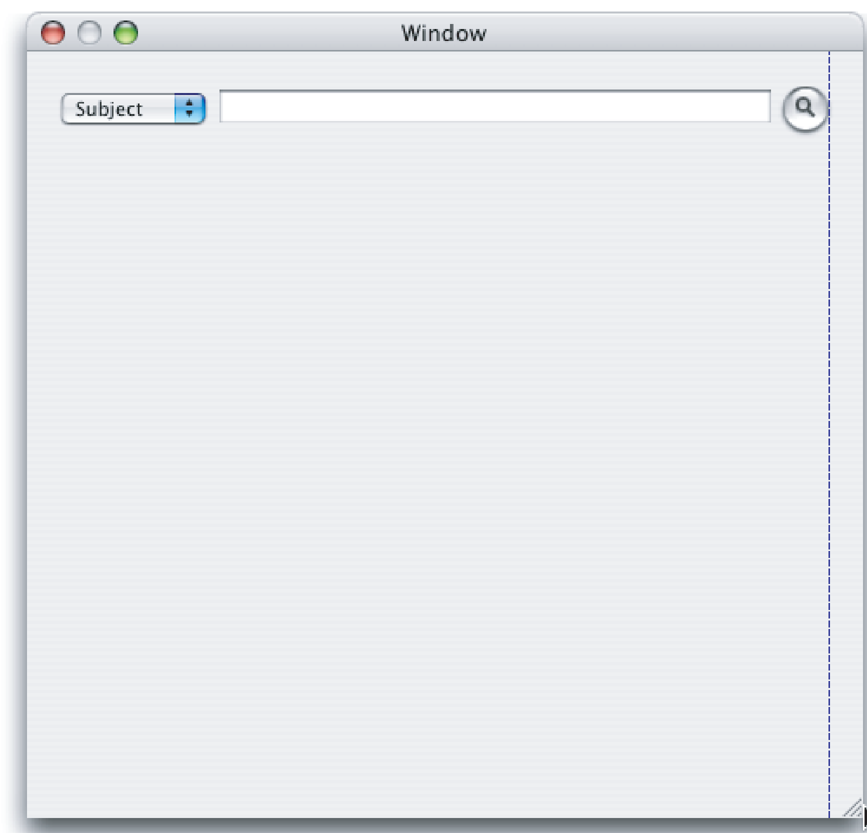
To center the icon, click the middle button in the top row under "Icon Position" in the Info window shown in Figure 7-37. The result is shown in Figure 7-38.

Figure 7-38 The button and magnifying glass



4. You have a couple of options for correctly positioning the button in relation to the right side of the window. You can drag the button to the right and use Interface Builder's alignment guides to align it with the edge of the window, then expand the text field toward the button until it is also aligned. Or you can resize the window itself, moving the right edge toward the button, again using Interface Builder's alignment guides to align the window edge. Figure 7-39 shows the latter process.

Figure 7-39 Aligning the right edge of the window with the button



2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Set Up an Outline View

Mail Search uses an outline view to show the Mail folders that can be searched. It can be difficult to determine in Interface Builder whether you have selected an outline view or the scroll view that contains it. Single-clicking selects the containing scroll view, while double-clicking selects the outline view. The tutorial steps that follow describe visual cues to determine which view is selected.

> **Note:** If you have not already experimented with AppleScript Studio's Outline sample application, you should consider doing so before performing the steps in this section.

To be certain whether an outline view or the containing scroll view is selected, you can also open an Info window to the AppleScript pane. When that pane is displayed, the window title for the Info window shows the class of the selected item: either NSScrollView if the scroll view is selected, or NSOutlineView if the outline view is selected (or NSTableView when a table view is selected).

As a final alternative, you can display the objects in a nib window in outline view (as described in "Examining an Object Hierarchy in the Nib View" (page 182)), find the object (either an outline view, a scroll view, or any other object) in the view hierarchy, and double-click the object to select it in its window and to display it in the Info window.

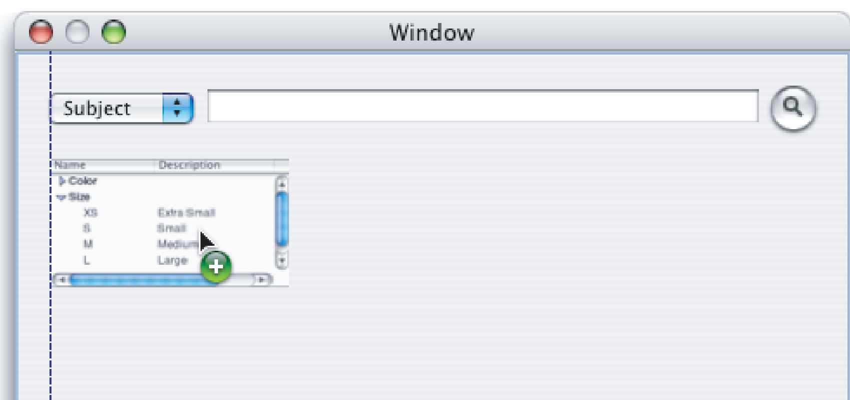
To set up an outline view in the search window, perform these steps:

1. Drag an outline view from the Cocoa-Data palette in the Palette window to the search window. The outline view is in the upper-left corner with "Name" and "Description" columns.

   Use the automatic feedback to position the outline view in the top left corner of the search window, below the previously placed popup button. Align it with the left side of the window, as shown in Figure 7-40.

   When you release the outline view, it takes on a more generic format, as shown (after resizing) in Figure 7-41.

Figure 7-40 Inserting an outline view in the search window

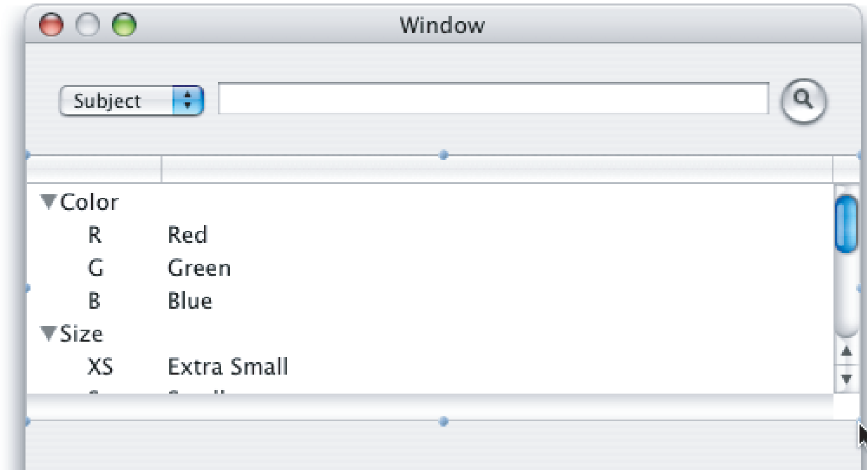


2. To resize the outline view, click to select it, then drag the lower-right corner. The text field should stretch all the way across the window and about half the way down (leaving room for a table view to display the search results), as shown in Figure 7-41.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Note: The outline view text visible in Figure 7-41 (for colors and sizes) is simply filler text displayed by Interface Builder to help you see the rows and columns in the outline view. It will not be visible when you build the application.

Figure 7-41 Resizing the outline view in the search window



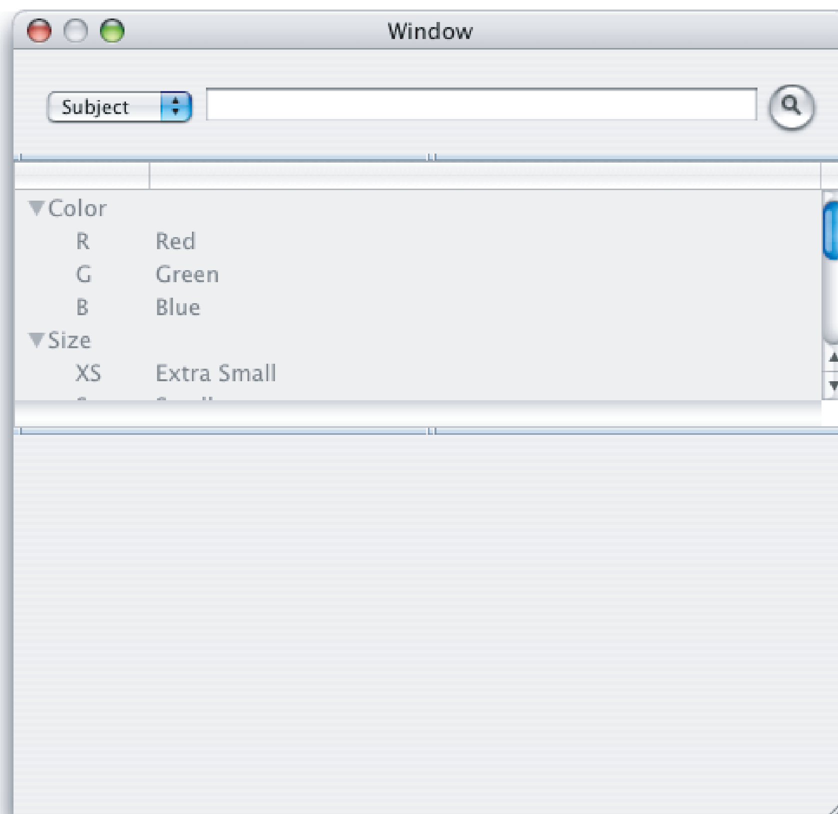
3. By default, a new outline view has two columns, but you only need one column to display mailboxes. To change the number of columns, double-click the outline view. The result is shown in Figure 7-42.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Important: If you single-click an outline view in Interface Builder, you will see the outline view object name in the Info window, but you have only selected the scroll view that is part of the outline view. You know the outline view is selected when it takes on the appearance shown in Figure 7-42, with the whole view showing the selection color.

You must also double-click to select a table view or other data view object. See also the section "Examining an Object Hierarchy in the Nib View" (page 182).

Figure 7-42 A selected outline view



Now open the Info window to the Attributes pane. The result is shown in Figure 7-43. Make the following changes to the default settings:

a. In the Allow section, select the Multiple Selection checkbox.

b. In the Display section, deselect Horizontal Scroller.

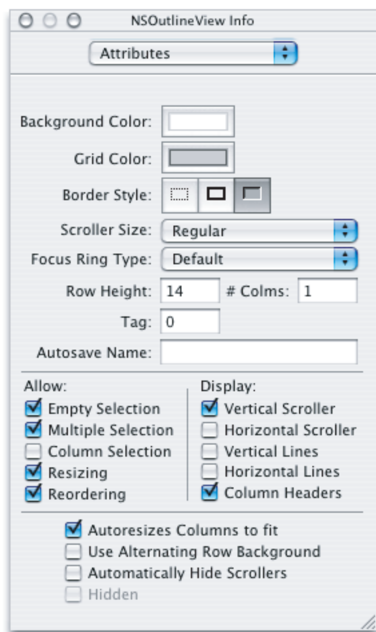
c. In the Options section, select the "Autoresizes Columns to fit" checkbox; deselect the Allows Reordering checkbox.

d. Set the row height to 14.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

e. Set the number of columns to 1.

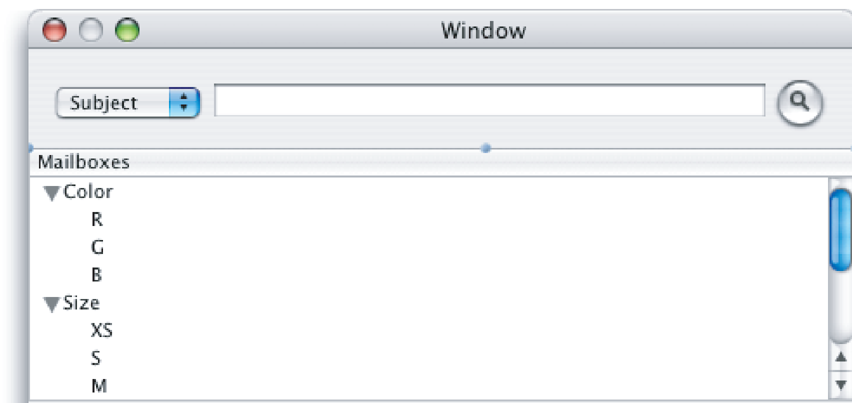
Figure 7-43 The Info window for the outline view, after changes



4. While you still have the Info window open for the outline view, display the AppleScript pane, then type "mailboxes" in the Name field to supply an AppleScript name for the outline view.

5. To add a column title to the outline view, double-click the view to select it, then double-click the title row at the top. You can then type "Mailboxes" for the column title. The result is shown in Figure 7-44.

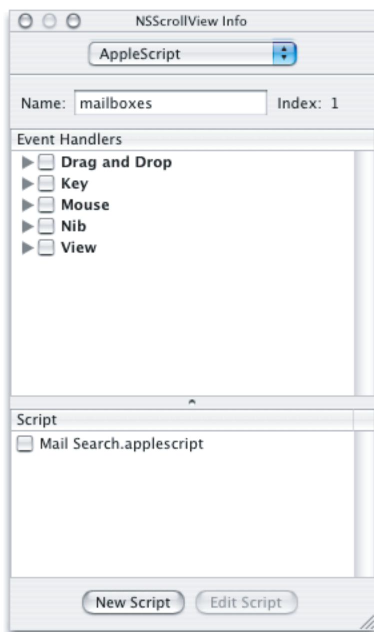
Figure 7-44 The outline view after naming the Mailboxes column



2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

6. Mail Search also requires an AppleScript name for the scroll view that contains the outline view. To provide this name, single-click the outline view, open the Info window (for the scroll view) to the AppleScript pane, then type "mailboxes" in the Name field. The result is shown in Figure 7-45.

Figure 7-45 The Info window for the scroll view containing the outline view



7. If you haven't saved the Document.nib file recently, do so now.

Set Up a Table View

Mail Search uses a table view to show search results—the messages that contain the search text.

Note: If you have not already experimented with AppleScript Studio's Table sample application, you should consider doing so before performing the steps in this section. The version that uses a data source object is recommended.

To set up a table view in the search window, perform these steps:

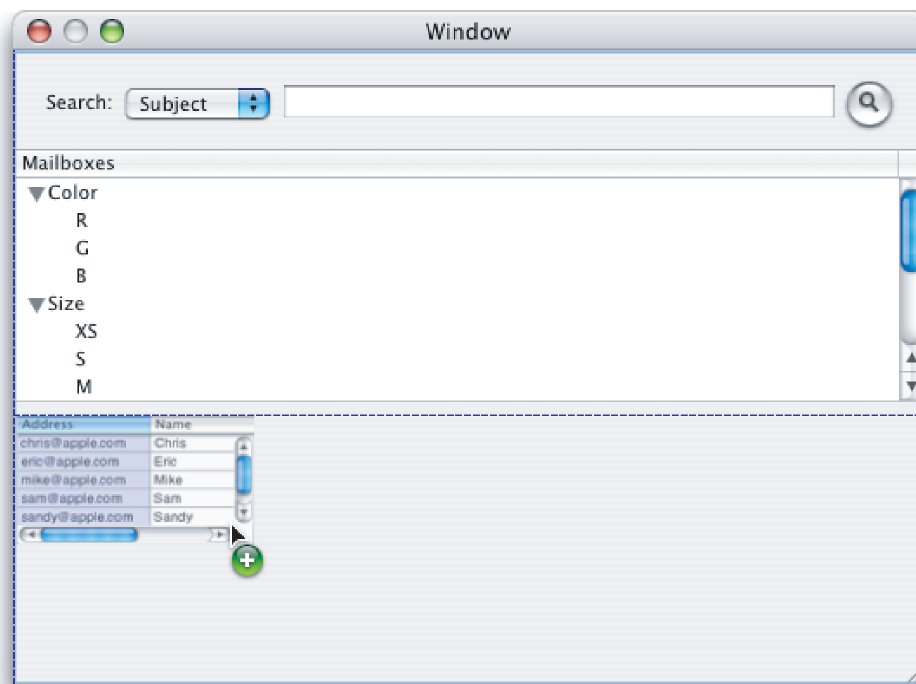
1. Drag a table view from the Cocoa-Data palette in the Palette window to the search window. The table view is in the lower-left corner with "Address" and "Name" columns.

   Use the automatic feedback to position the table view below the previously placed outline view. Align it with the left side of the window, as shown in Figure 7-46.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

When you release the table view, it takes on a more generic format.

Figure 7-46 Inserting a table view in the search window



2. To resize the table view, click to select it, then drag the lower-right corner. The text field should stretch all the way across the window and all the way to the bottom.

3. By default, a new table view has two columns, but you need three columns (for From, Subject, and Mailbox). Use the same steps described in "Set Up an Outline View" (page 159) to open the Info window for the table view. Make the following changes to the default settings:

    a. In the Display section, deselect the Horizontal Scroller checkbox.

    b. In the lower options section, select the "Autoresizes Columns to fit" checkbox.

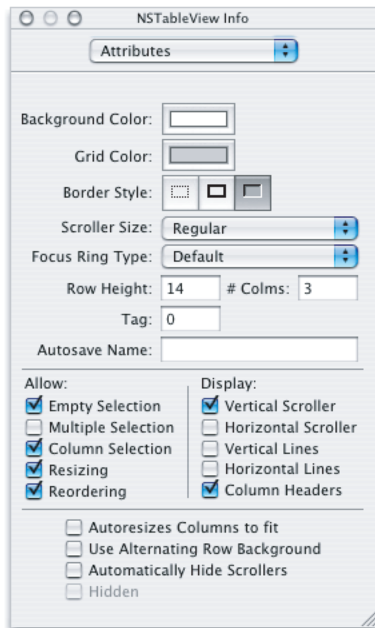
    c. Set the row height to 14.

    d. Set the number of columns to 3.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

The resulting Info window is shown in Figure 7-47.

Figure 7-47 The Info window for the table view, after changes



4. While you still have the Info window open for the table view, display the AppleScript pane, then type “messages” in the Name field to supply an AppleScript name for the table view.

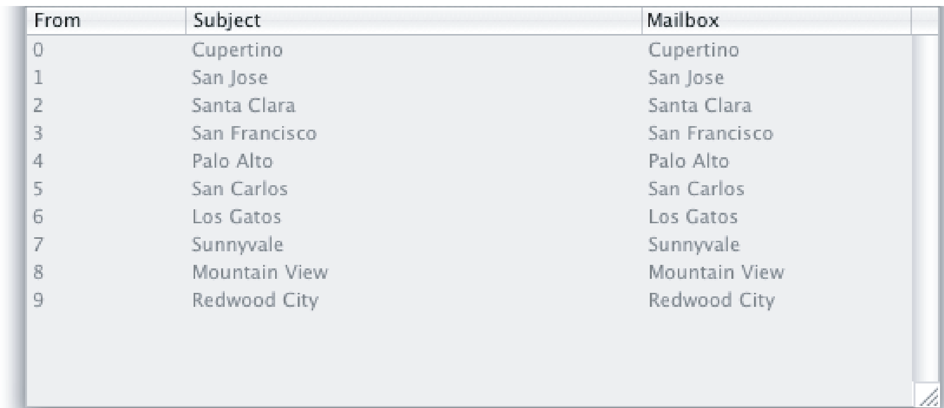
5. To resize a column in the table view, double-click the view to select it, then position the cursor over the column divider in the title row. The cursor changes to indicate when you can resize. Simply click and drag to resize a column.

   Start by moving the right-most column divider to the left, which reduces the size of the middle column. Then move the left-most divider to the right, increasing the size of the left column. Adjust the column sizes as needed so that the Subject and Mailbox columns are wider than the From column.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

6. To add column titles to the table view, use the same steps described in "Set Up an Outline View" (page 159). You can tab from column title to column title as you add titles. The result is shown in Figure 7-48.

Figure 7-48 The table view with titles



7. Mail Search also requires an AppleScript name for the scroll view that contains the table view. To name the scroll view, follow the same steps you used in "Set Up an Outline View" (page 159):

   a. Single-click the table view to select the scroll view that contains it.

   b. In the AppleScript pane of the Info window for the scroll view, type "messages" in the Name field.

8. If you haven't saved the Document.nib file recently, do so now.

Group the Outline View and Table View in a Split View

Mail Search uses a split view to combine the outline and table views you've previously installed. The split view allows a user to adjust the size of either view, depending on how much space they want to devote to the available mailboxes (in the outline view) and how much to the found messages (in the table view).

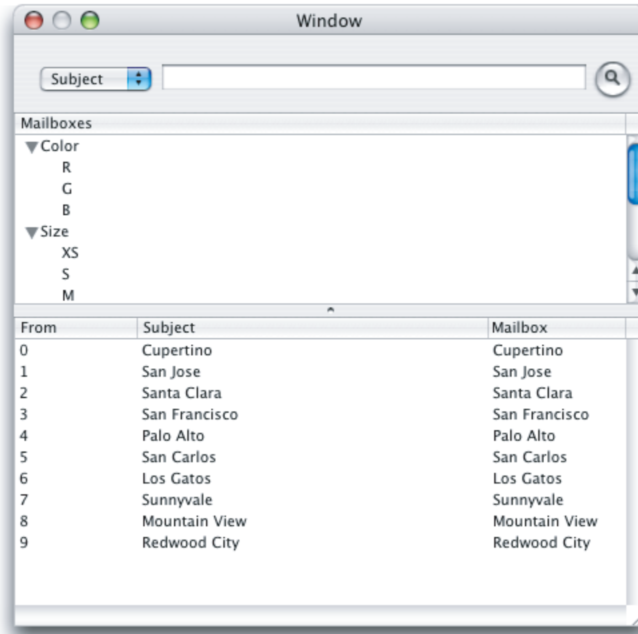
To group the outline view and table view in a split view, perform these steps:

1. In the search window you have been working on, click to select the table view. The view should have selection handles (small dots) on all sides, and you should see "NSTableView Info" as the title of the Info window. Shift-click to also select the outline view. The outline view should have selection handles and the Info window should display "Multiple Selection" in the Attributes pane.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

2. In the Layout menu, choose Split View in the "Make subviews of" submenu to make these two views into subviews of a split view. The result is shown in Figure 7-49. The distinguishing feature of a split view is the handle for modifying the relative sizes of the contained outline and table views.

Figure 7-49 The final search window, now containing a split view



3. Mail Search also requires an AppleScript name for the split view that contains the outline and table views. To name the split view, you follow similar steps to those listed in "Set Up an Outline View" (page 159):

   a. Single-click to select the split view. You may have to click on another object, then click the split view to select it.

   b. In the AppleScript pane of the Info window for the split view, type "splitter" in the Name field.

> **Important:** Once you have grouped the outline and table views in a split view, it takes more effort to select the table or outline view. For example, to select the Mailboxes outline view when the window is currently selected, you click once to select the split view, double-click to select the scroll view that contains the outline view (which causes it to have a thin black outline), and double-click again to select the outline view (causing the whole view to take on the current selection color).
>
> For more detail on selecting subviews, see "Examining an Object Hierarchy in the Nib View" (page 182).

4. Congratulations—you've completed Mail Search's user interface. If you haven't saved recently, do so now.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Mail Search Tutorial: Connect the Interface

Mail Search is a fairly complex AppleScript Studio application that searches for specified text in messages in the Mac OS X Mail application. In previous chapters you've designed the application, created a project, and built the user interface with the Interface Builder application. In this chapter, you connect objects in the interface to handlers in the application's script file.

This chapter assumes you have completed previous Mail Search tutorial chapters.

Connect the Interface

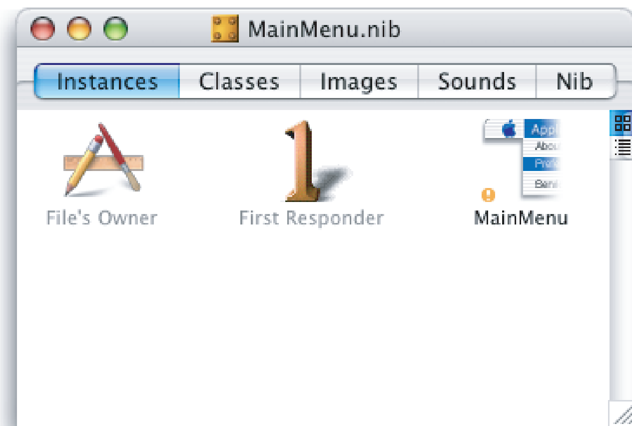
Now that you've built Mail Search's interface, you need to perform additional steps in Interface Builder to connect objects in the interface to event handlers in the project. In "Event Handlers in Mail Search" (page 128), you identified handlers that would be needed for various objects. You don't need any handlers for the status dialog or the message window, but you do need to make connections for Mail Search's application object and search window. For the search window, you also need to connect data source objects to provide data to the search results and mailbox views. (A data source object, described in more detail below, is a special object supplied by AppleScript Studio that provides row and column data to a table or outline view.) The next sections describe how to make these connections.

Connect the Application Object

The application object in an AppleScript Studio application is represented by the File's Owner object in the Instances tab in the MainMenu.nib window. This object is described in "Interface Creation" (page 54) and shown in Figure 8-1. Your application can use this object to connect handlers that are called at various interesting times, such as when the application is launched or activated. For a look at the available handlers, see Figure 8-2.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Figure 8-1 The File's Owner instance in the MainMenu.nib window



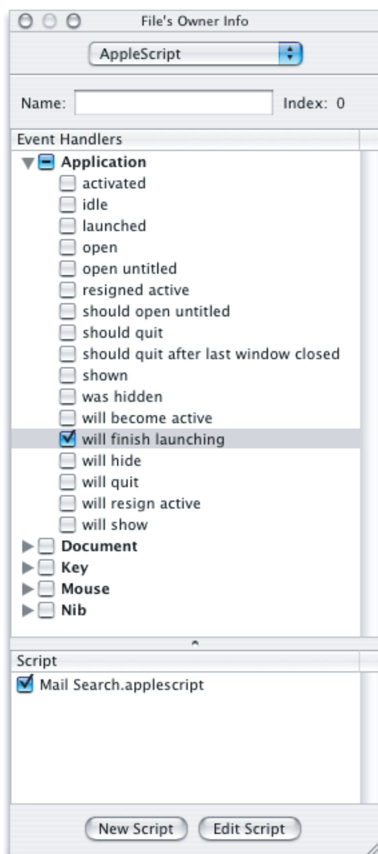
Mail Search uses the `will finish launching` handler to perform necessary initialization after the application has created its interface but has not yet completed launching. To connect this handler, perform the following steps:

1. Select the File's Owner object in the Instances tab in the MainMenu.nib window, as shown in Figure 8-1.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

2. Choose Show Info from the Tools menu or type Command-Shift-I to open the Info window. Use the pop-up menu at the top of the window to display the AppleScript pane. Then click the disclosure triangle to reveal the handlers in the Application group. The result is shown in Figure 8-2.

Figure 8-2 The Info window for the File's Owner instance



3. To connect the handler, perform the following steps:

   a. Select the `will finish launching` checkbox.

   b. Select the file `Mail Search.applescript` in the Script list.

   c. Click the Edit Script button.

When you click the Edit Script button, Interface Builder inserts an empty `will finish launching` handler in the script file and opens the file in an Xcode editor window. The handler is shown in Listing 8-1. You add statements to this handler in "Application Object Handler" (page 189).

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Note: All event handlers start with the keyword `on`. Handlers can have zero or more parameters. The `will finish launching` handler has one parameter, `theObject`, which represents the user interface object that received the `clicked` message. Most event handlers in AppleScript Studio have this same parameter.

Listing 8-1 A new handler declaration for the `will finish launching` handler

```
on will finish launching theObject
    (*Add your script here.*)
end will finish launching
```

4. When you create a new event handler for an object, you should also insert a comment that specifies which object initiates calls to the handler. In this case, you could add the line `(* "Called from the application object just prior to completion of launching." *)`. In some cases, more than one object of the same type may be connected to the handler. This situation is described in "Deciding How Many Script Files to Use" (page 62).

5. To verify that the handler is correctly connected, you can replace the comment `(*Add your script here.*)` with `display dialog "In the will finish launching handler."`

   You can build and run the application by typing Command-R, choosing Build and Run from the Build menu, or clicking the Build and Run button. You should see the dialog after the application launches.

   As you work through additional sections that connect handlers, you can use the same techniques to verify that the handlers are working.

Important: After editing a script file in Xcode, you should always save it before returning to Interface Builder to add more connections.

6. As always, save the nib file after completing your changes.

As noted in "Add the Message Window to the Nib File" (page 139), the first time you make a change in the AppleScript pane, Interface Builder adds a new instance named AppleScript Info to the nib window.

Connect Interface Items in the Search Window

The search window is the main place where you connect Mail Search's interface to event handlers. You also need to connect data source objects to provide data to the search results and mailbox views. To make these connections, perform these steps, which are described in detail in the sections that follow:

1. Connect event handlers to the search window.

2. Connect an event handler to the text field.

3. Connect an event handler to the find button.

4. Connect an event handler to the search results view.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

5. Connect data source objects to the mailboxes view and the search results view.

Connect the Search Window

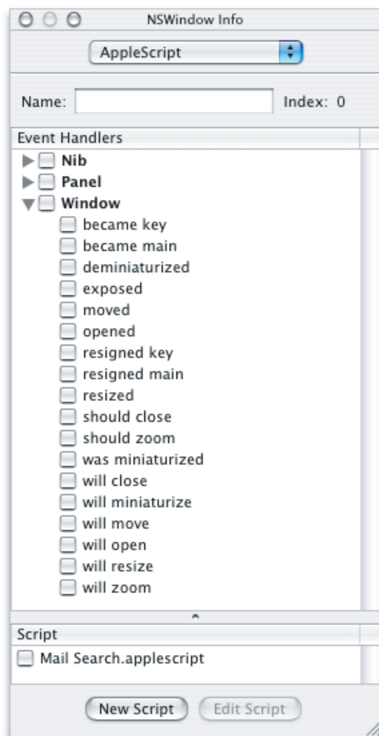
As described in "Plan the Code" (page 127), Mail Search needs to know when a search window is opened, activated, or closed. To add these handlers, perform the following steps:

1. If the Document.nib file is not already open in Interface Builder, open it.

2. Select the Mail Search instance in the nib window.

3. Choose Show Info from the Tools menu or type Command-Shift-I to open the Info window. Use the pop-up menu at the top of the window to display the AppleScript pane. Then click the disclosure triangle to reveal the handlers in the Window group. The result is shown in Figure 8-3.

Figure 8-3 The Info window for the Mail Search window instance



4. To connect the handlers, perform the following steps:

   a. Select the checkboxes for `became main`, `will close`, and `will open`.

   b. To connect the handlers, select the checkbox for the script file `Mail Search.applescript` in the Script list.

   c. Click the Edit Script button.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

When you click the Edit Script button, Interface Builder inserts empty handlers for the three event handlers in the script file and opens the file in an Xcode editor window. The handlers are shown in Listing 8-2. You add statements to these handlers in "Search Window Handlers" (page 189).

Listing 8-2 New handler declarations for several handlers

```
on became main theObject
    (*Add your script here.*)
end became main

on will close theObject
    (*Add your script here.*)
end will close

on will open theObject
    (*Add your script here.*)
end will open
```

5. To verify that the handlers are correctly connected, you can add `display dialog` statements, as described in "Connect the Application Object" (page 169). You may also wish to add comments that the `theObject` parameter will be a search window object.

Connect the Text Field

A user types text to search for in the text field in the search window. In this section, you connect an action handler to the text field so that when a user presses the Enter key, Mail Search initiates a search. To add this handler, perform the following steps:

1. In Interface Builder, open the Info window for the Mail Search window instance and display the AppleScript pane, as described in "Connect the Application Object" (page 169).

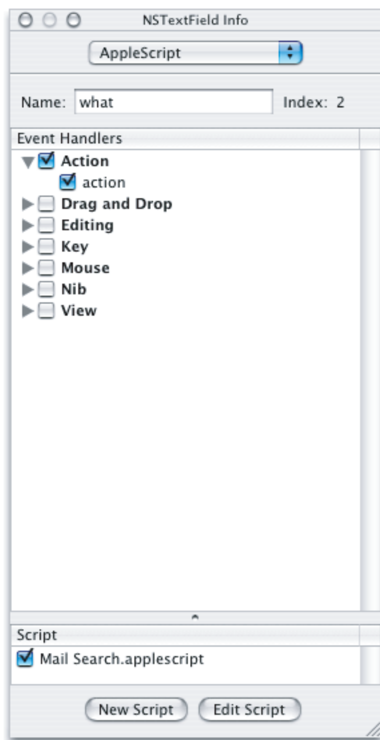
2. Select the text field in the search window.

3. Click to open the Action group, then select the checkbox for the `action` handler. The result is shown in Figure 8-4.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Action handlers get their name from a Cocoa concept known as target-action. In this case, pressing the return key can cause a text field object to send an action message to its target (in Cocoa, the target is a method, but here it is a handler). Other examples of action events are `clicked` and `double-clicked`.

Figure 8-4 The Info window for the search text field



4. Connect the handler as in previous sections by selecting the file `Mail Search.applescript` in the Script list, then click the Edit Script button to open the script in an Xcode editor window.

   The resulting handler declaration is shown in Listing 8-3. You add statements to this handler in "Text Field Handler" (page 191).

Listing 8-3 A new `action` handler for a text field

```
on action theObject
    (*Add your script here.*)
end action
```

5. To verify that the handler is correctly connected, you can add `display dialog` statements, as described in "Connect the Application Object" (page 169). You may also wish to add a comment that the `theObject` parameter will be a text view object.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

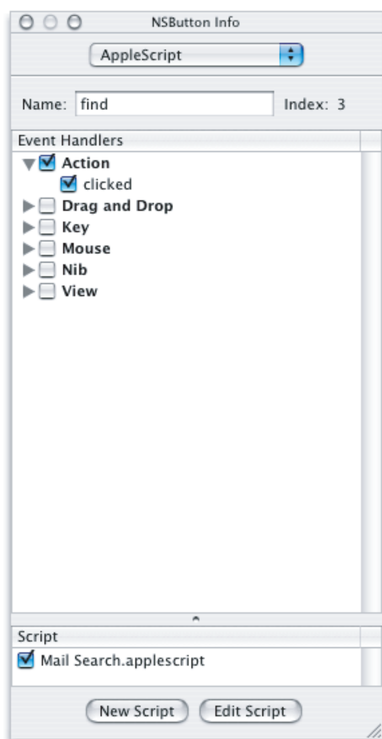
Connect the Find Button

A user clicks the find button in Mail Search's search window to initiate a search. To enable this behavior, you connect a `clicked` handler for the find button. To do so, perform steps similar to those you've used in previous sections:

- select the find button
- open the Info window
- display the AppleScript pane
- select the checkbox for the `clicked` handler
- To connect the handlers, select the checkbox for the script file `Mail Search.applescript`; the result is shown in Figure 8-5.
- click the Edit Script button to insert a `clicked` handler declaration in the script file

You can examine the resulting handler declaration in Xcode. You add statements to the handler in "Find Button Handler" (page 191).

Figure 8-5 The Info window for the find button



Connect the Search Results View

Mail Search uses a table view to show search results—the found messages that contain the search text. When a user double-clicks a found message in the table view, Mail Search displays the message in a separate window. To support this behavior, the table view object needs a `double-clicked` handler.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

To connect a `double-clicked` handler for the table view in the search window, perform the same steps you've used in previous sections:

- click, then double-click (and double-click again if necessary) to select the table view
- open the Info window
- display the AppleScript pane
- select the checkbox for the `double-clicked` handler
- select the checkbox for the script file `Mail Search.applescript`
- click the Edit Script button to insert a `double-clicked` handler declaration in the script file

The result before clicking the Edit Script button is shown in Figure 8-6.

In Figure 8-6, the Action group and clicked checkboxes have dashes while the double clicked checkbox has a checkmark. A handler checkbox (such as the double clicked checkbox) can have one of three states:

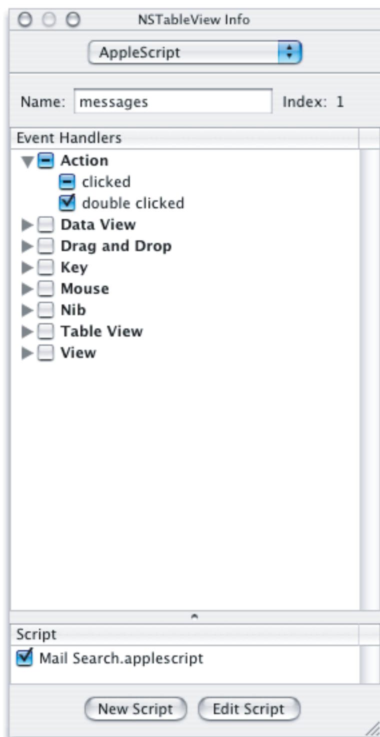
- Empty: Indicates there is no handler with that name in the currently selected script.
- Dashed: Indicates that there is an event handler with that name in the currently selected script, but that the current object isn't connected to it (so presumably some other object is).
- Checked: Indicates there is an event handler with that name in the currently selected script (or that such a handler will be added when you edit the script), and it is (or will be) connected to the current object in the Info window.

A group checkbox (such as the Action group checkbox) can have one of the same three states:

- Empty: Indicates no event handler in that group is checked.
- Dashed: Indicates at least one event handler in that group is checked.
- Checked: Indicates every event handler in that group is checked.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Figure 8-6 The Info window for the search results table view



You can examine the resulting `double-clicked` handler declaration in Xcode. You add statements to the handler in "Search Results Table View Handler" (page 192).

Provide a Data Source Object for the Mailboxes View

Mail Search uses an outline view to display a list of all mailboxes from all accounts. An outline view can display hierarchical data, similarly to the way the Finder displays directories and files in a list view. The outline view needs to be connected to a **data source object**—a special object supplied by AppleScript Studio that provides row data (in this case, mailbox names) to a table or outline view.

> **Note:** See the Older Release Notes in Xcode for information on how to create a data source and hook it up with two lines in your script file, rather than following the steps listed here.

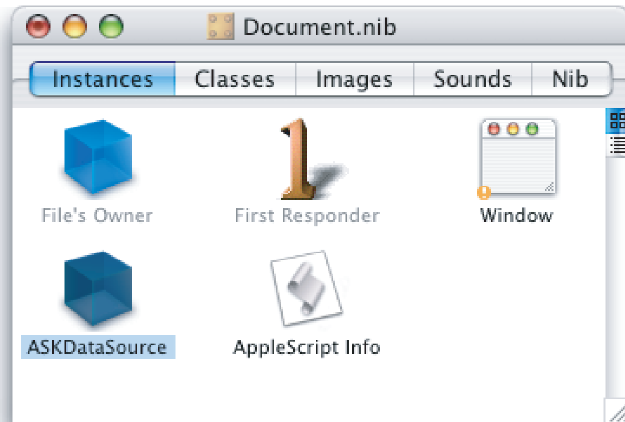
To create a data source object and connect it to the outline view in the search window, perform these steps (assuming you still have the search window from the `Document.nib` file open in Interface Builder, along with the Info window):

1. Click the AppleScript button in the Palette window toolbar to select the AppleScript pane. There currently is a single image in that pane, representing a data source object.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

2. Drag a data source object from the Palette window to the Document.nib window. The result is shown in Figure 8-7. The letters "ASK" come from AppleScriptKit, the framework that defines this data source.

Figure 8-7 The Document.nib window with a data source object



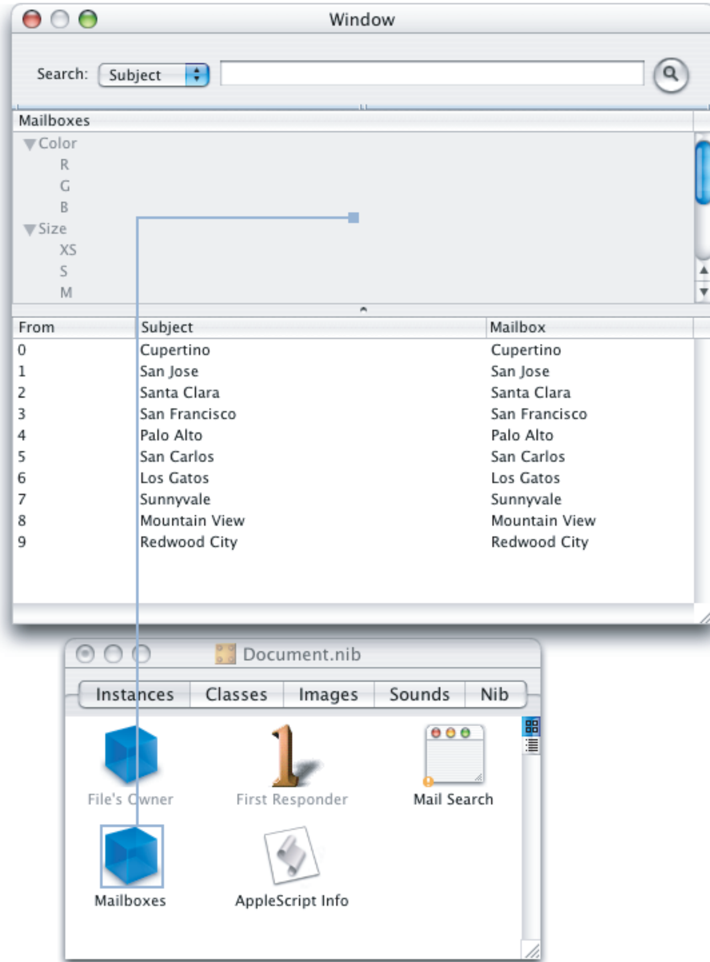
3. Double-click the name ("ASKDataSource") and change it to "Mailboxes".

4. Double-click to select the outline view in the search window. Be sure the view appears as in Figure 7-42 (page 161), so that you know it's selected.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

5. Press and hold the Control key and drag from the outline view to the data source object in the Document.nib window. This step is shown in Figure 8-8.

Figure 8-8 Connecting the outline view to the data source object

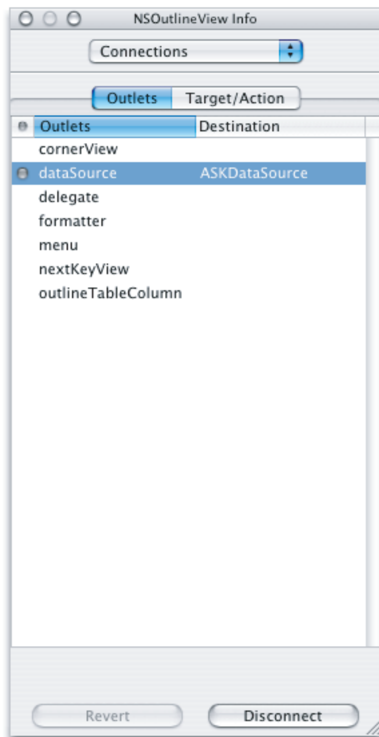


6. At this point, the Info window displays the Connections pane. Double-click the dataSource outlet. This connects the outline view to the data source object. The result is shown in Figure 8-9.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Even though you named the data source “Mailboxes,” the destination of the connection in Figure 8-9 is still “ASKDataSource.”

Figure 8-9 The Info window for the outline view after connecting a data source outlet



7. Mail Search also needs an identifier for the column header (which has the column title Mailboxes) in the outline view. Data source objects use identifiers specify columns and rows, since they can be shuffled. To provide an identifier, perform these steps:

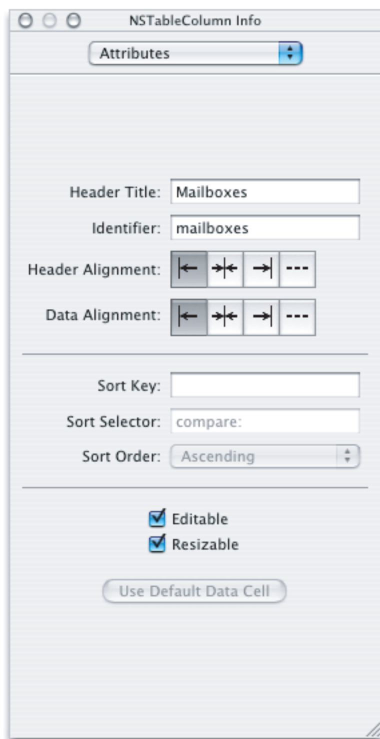
   a. With the outline view still selected in the search window, click to select the column header.

   b. Choose the Attributes pane in the Info window.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

c. Type “mailboxes” in the Identifier text field. The result is shown in Figure 8-10.

Figure 8-10 The Info window after entering an outline column identifier



8. Don’t forget to periodically save the nib file.

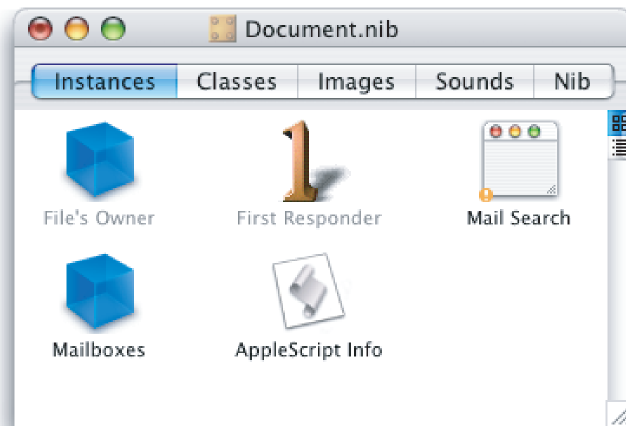
Examining an Object Hierarchy in the Nib View

In “Group the Outline View and Table View in a Split View” (page 166), you saw that once you have a series of deeply nested views, it can become difficult to select one of them. However, Interface Builder provides a mechanism to make it easier to display view hierarchies and connections, and to select nested objects. That mechanism consists of displaying the nib window in outline view. In this case, “outline view” refers to a mode of window display, not an outline view object.

Figure 8-11 shows the Document.nib window after connecting a data source for the outline view object.

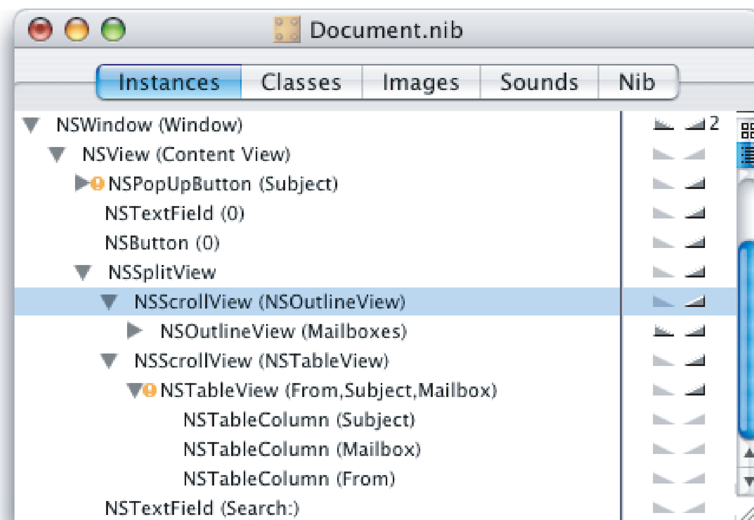
2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Figure 8-11 The Document.nib window



By clicking the small outline view icon on the right-hand side of the view, you can display contents of the window in outline view, as shown in Figure 8-12.

Figure 8-12 The Document.nib window in outline view

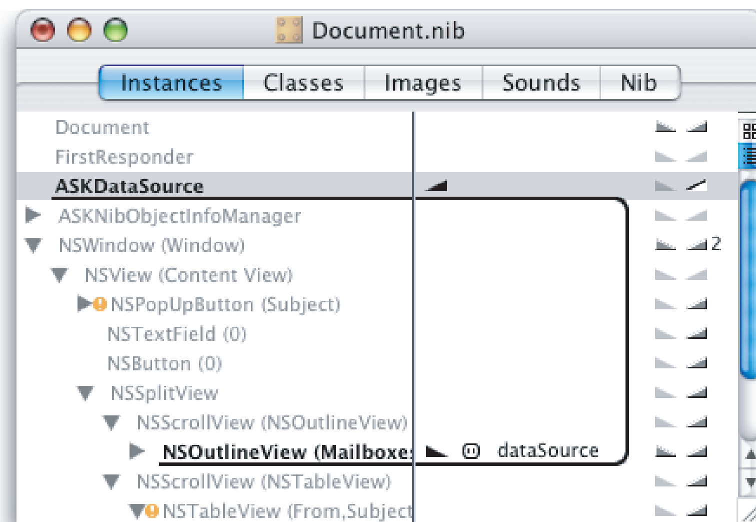


In this case, the window shows the expanded hierarchy for the search window. Instead of interface objects (such as buttons), it shows the Cocoa classes for the objects (NSButton). You can view the hierarchy for any object in the nib window. Double-clicking an object (such as NSOutlineView (Mailboxes)) in the Instances tab selects the object in its window, providing a simple way to select a nested item such as an outline view within a scroller within a split view. If the Info window is open, it also displays the object in the Info window.

To examine the connections for an object, you click the small triangles next to it in the right-hand column. For example, Figure 8-13 shows the connections for the NSOutlineView object (the mailboxes outline view).

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Figure 8-13 Connections for the NSOutlineView object



Provide a Data Source Object for the Search Results View

Mail Search uses a table view to display the messages found by a search. The table includes columns for the message sender, the message subject, and the message mailbox. The table view needs to be connected to a data source object that provides row data to the table view.

To create a data source object and connect it to the table view in the search window, perform the same series of steps you used to connect a data source to the outline view in "Provide a Data Source Object for the Mailboxes View" (page 178). Assuming you still have the search window from the `Document.nib` file open in Interface Builder, along with the Info window, the steps include:

1. Drag a data source object from the AppleScript pane in the Palette window to the Document.nib window.

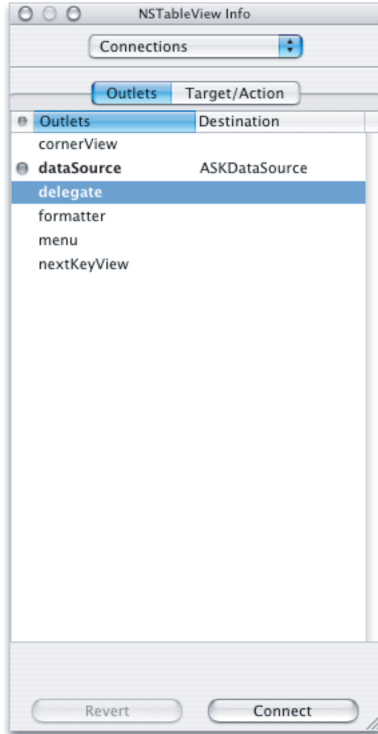
2. Change the name of the data source object from "ASKDataSource" to "Messages".

3. Select the table view in the search window, then press and hold the Control key and drag from the table view to the Messages data source object in the Document.nib window.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

4. Double-click the dataSource outlet in the Connections pane of the Info window. to connect the table view to the data source object. The result is shown in Figure 8-14.

Figure 8-14 The Info window for the table view after connecting a data source outlet



5. You also need to provide an identifier for each column header (the column headers are titled From, Subject, and Mailbox) in the table view. To do so, perform these steps:

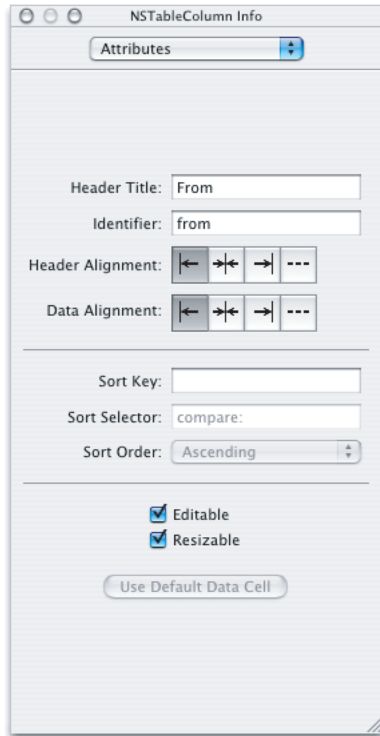
   a. With the table view still selected in the search window, click to select the column header.

   b. Choose the Attributes pane in the Info window.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

c. Type the name (either "from," "subject," or "mailbox") in the Identifier text field. Save your results after typing each new identifier. The result for the From column header is shown in Figure 8-15.

Figure 8-15 The Info window after entering a table column identifier



6. As always, save the Document nib file after completing your changes.

You have now completed connecting the interface for the Mail Search application.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Mail Search Tutorial: Write the Code

In this chapter you'll write the scripts and handlers for Mail Search, an AppleScript Studio application that searches for specified text in messages in the Mac OS X Mail application. You'll also build and test the completed application. To do this, you'll perform the steps described in the following sections:

1. "Define Global Variables and Properties" (page 188).
2. "Write Event Handlers for the Interface" (page 188).
3. "Write Scripts and Additional Handlers" (page 192).

This chapter assumes you have completed previous Mail Search tutorial chapters.

Obtaining the Code for the Mail Search Tutorial

The Mail Search tutorial chapters in this document contain listings for most of the handlers and script objects the application uses, and you can find a complete listing in "Chapter 10" (page 187). However, both for convenience and completeness, it is recommended that you obtain script statements by copying them from the Mail Search sample application distributed with AppleScript Studio.

> **Note:** The scripting terminology for the Mail application changed with Mac OS X version 10.2. The scripts and examples created prior to that version differ from their counterparts today. The listings shown in this document reflect all revisions through Mac OS X version 10.3.

Look for the file `Mail Search.applescript` in the Mail Search project folder. See "AppleScript Studio Sample Applications" (page 35) for information on where to find the sample applications.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Note: If you double-click `Mail Search.applescript` in the Finder, the Finder will try to open it with the Script Editor application. You can open the file from the Files list in the Groups & Files list in the Mail Search sample application. Or, from your tutorial Mail Search project, you can choose Open from the File menu and navigate to `Mail Search.applescript` in the Mail Search sample application folder.

When you open this file for the first time in the Mail Search application, it may launch the Mail application, if it isn't already running.

You should also read the section "Switching Between AppleScript Studio and Script Editor" (page 70) before beginning this tutorial.

Define Global Variables and Properties

In "Plan the Code" (page 127), you identified a handful of global variables and properties Mail Search makes available to all its scripts and handlers. The one global variable is a global list to keep track of instances of the controller script. Each controller script handles search operations for one search window. You define this global variable as shown in Listing 9-1 and place it at the top of the script file `Mail Search.applescript`.

Listing 9-1 Global list variable to store instances of controller script

```
(*==== Globals ====*)

global controllers
```

Mail Search uses two global properties, a counter to keep track of the number of open windows and a boolean to keep track of whether the status panel nib file has been opened. You define these properties as shown in Listing 9-2 and place them at the top of the script file `Mail Search.applescript`, below the global variable you just defined.

Listing 9-2 Properties used in Mail Search

```
(*==== Properties ====*)

property windowCount: 0
property statusPanelNibLoaded: false
```

Write Event Handlers for the Interface

In "Plan the Code" (page 127), you identified event handlers for objects in Mail Search's interface that must respond to user actions or changes in application state (pending or completed). For example, the search window has three handlers: `became main`, called when the window becomes active; `will open`, called as the window is about to open, and `will close`, called when the window is about to close. The find button has one handler, `clicked`, called when a user clicks the button.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

In "Connect the Interface" (page 169), you hooked up objects in the interface to the required handlers. As part of that step, Interface Builder inserted event handler declarations in the file `Mail Search.applescript`. In this section, you'll write script statements for those handlers.

When planning the code, you also specified additional scripts and handlers to carry out operations such as searching, displaying results, and displaying status. You'll write those scripts and handlers in "Write Scripts and Additional Handlers" (page 192). The handlers you'll write in this section don't do much except call other handlers, so you won't be able to build and run the application until you've completed both sections (though you can compile `Mail Search.applescript` to check syntax).

While this approach may be suitable for simple applications—or for a tutorial, where the end result is already known—it's not recommended for complex, real-world applications. For those applications, you will most likely work incrementally, adding parts of the interface, connecting them to event handlers, and testing individual scripts and handlers as you create them.

As a workaround to allow you to continue to build the application, you can add empty versions of the handlers you haven't implemented yet to the script file. And as always, you can put `display dialog` statements in the stubbed out handlers to show when they are called.

Position each of the event handlers described here at the beginning of the script file `Mail Search.applescript`, just after the global variables and properties defined earlier.

Before working on this section, be sure you've read the section "Obtaining the Code for the Mail Search Tutorial" (page 187).

Application Object Handler

You previously connected a `will finish launching` handler for the application object. That handler is called after the application's user interface has been unarchived from its nib files and just before the application enters its main event loop. In that handler, Mail Search can do any additional initialization it requires before a user performs any actions.

The only initialization Mail Search requires is to set the global `controllers` variable to an empty list, as shown in Listing 9-3.

Listing 9-3 The `will finish launching` handler for the application object

```
on will finish launching theObject
    set controllers to {}
end will finish launching
```

Search Window Handlers

You previously connected three handlers for the application.search window: `will open`, `became main`, and `will close`.

The `will open` handler is called after a window has been created from a nib file and before the window opens. At this point, Mail Search can do any additional initialization for the window. The handler is shown in Listing 9-4.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Listing 9-4 `will open` handler for search window

```
on will open theObject
    set theController to makeController(theObject)
    if theController is not equal to null then
        addController(theController)
        tell theController to initialize()
    end if
end will open
```

The `will open` handler performs these operations:

1. It calls `makeController` to create a controller for the window. The controller is a script that responds to user actions in the window.

2. If `makeController` successfully returns a controller, the `will open` handler adds the controller to the global list of controllers, then tells the controller to initialize itself.

The `became main` handler is called when a window becomes the current window. It's similar to what you might be familiar with as an activate event—an event a window receives when it needs to present itself as the active window. The handler is shown in Listing 9-5.

Listing 9-5 The `became main` handler for the search window

```
on became main theObject
    set theController to controllerForWindow(theObject)
    if theController is not equal to null then
        tell theController to loadMailboxes()
    end if
end became main
```

The `became main` handler performs these operations:

1. It calls `controllerForWindow` to get the controller for the window. The controller should have been created during a previous call to `will open`.

> **Note:** The `became main` handler passes `theObject` when it calls `controllerForWindow`. It does not have to pass `(window of theObject)`, as `action`, `clicked`, and `double clicked` handlers do, because `became main` is a handler for the window object and `theObject` is already a reference to the window object.

2. If `controllerForWindow` successfully returns a controller, the `became main` handler calls the controller's `loadMailboxes` handler (shown in Listing 9-12 (page 194)) to search the Mail application for all available mailboxes and display them in the Mailboxes outline view.

The `will close` handler is called before a window closes. At this point, Mail Search can do any cleanup for the window. The only cleanup Mail Search requires is to remove window's controller from the global list of controllers, as shown in Listing 9-6.

Listing 9-6 The `will close` handler for the search window

```
on will close theObject
    removeController(theObject)
```


2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

```
end will close
```

Text Field Handler

You previously connected an `action` handler for the search text field object. That handler is called when a user presses the Return key. The handler is shown in Listing 9-7.

Listing 9-7 The `action` handler for the search text field

```
on action theObject
    set theController to controllerForWindow(window of theObject)
    if theController is not equal to null then
        tell theController to find()
    end if
end action
```

The `action` handler performs these operations:

1. It calls `controllerForWindow` to get the controller for the window.

2. If `controllerForWindow` successfully returns a controller, the `action` handler calls the controller's `find` handler (shown in Listing 9-16 (page 197)) to search for the specified text in messages in the selected mailboxes (and display any matching messages in the Messages table view).

Find Button Handler

You previously connected a `clicked` handler for the find button object. That handler is called when a user clicks the find button. The handler is shown in Listing 9-8.

Listing 9-8 The `clicked` handler for the find button

```
on clicked theObject
    set theController to controllerForWindow(window of theObject)
    if theController is not equal to null then
        tell theController to find()
    end if
end clicked
```

The `clicked` handler is virtually identical to the `action` handler described in "Text Field Handler" (page 191):

1. It calls `controllerForWindow` to get the controller for the window.

2. If `controllerForWindow` successfully returns a controller, the `clicked` handler calls the controller's `find` handler (shown in Listing 9-16 (page 197)) to search for the specified text in messages in the selected mailboxes (and display any matching messages in the Messages table view).

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Search Results Table View Handler

You previously connected a `double clicked` handler for the Messages table view object. That handler is called when a user double-clicks a selected message in the table. The handler is shown in Listing 9-9.

Listing 9-9 The `double clicked` handler for the search results table view

```
on double clicked theObject
    set theController to controllerForWindow(window of theObject)
    if theController is not equal to null then
        tell theController to openMessages()
    end if
end double clicked
```

The `double clicked` handler performs these operations:

1. It calls `controllerForWindow` to get the controller for the window.
2. If `controllerForWindow` successfully returns a controller, the `double clicked` handler calls the controller's `openMessages` handler (shown in Listing 9-18 (page 201)) to open the selected message (or messages) in a separate window.

Write Scripts and Additional Handlers

In "Plan the Code" (page 127), you identified handlers for objects in Mail Search's interface that must respond to user actions or changes in application state (pending or completed). In "Connect the Interface" (page 169), you hooked up objects in the interface to the required handlers. Then you wrote the event handlers, in "Write Event Handlers for the Interface" (page 188).

When planning the code, you also specified additional scripts and handlers to carry out operations such as searching, displaying results, and displaying status. In this section, you'll write those scripts and handlers. As mentioned previously, you won't be able to build and run the application until you've completed this section, though you can compile `Mail Search.applescript` to check syntax.

Before working on this section, be sure you've read the section "Obtaining the Code for the Mail Search Tutorial" (page 187).

Write the Controller Script

Mail Search defines a controller script to perform tasks associated with the search window, including finding and displaying mailboxes, and finding and displaying messages that match the current search criteria. You specified the properties and handlers for this script in "The Controller Script" (page 129). In this section, you'll look at the actual script and the handlers it contains. The controller script is defined in the `makeController` handler. All of the controller's properties and handlers shown here are defined in that handler, which is listed in full in "Chapter 10" (page 187).

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Controller Script Properties and Initialization

The controller script is defined in the `makeController` handler, which is described in "Write Handlers for Working With Controllers" (page 203). The script defines and initializes properties for several things it needs to keep track of:

- `theWindow`: a reference to its window
- `theStatusPanel`: a reference to a status panel
- `foundMessages`: a list of found messages
- `mailboxesLoaded`: a boolean for whether it has created a list of available mailboxes

Listing 9-10 shows the definitions for these properties.

Listing 9-10 Properties of the controller script

```
property theWindow : forWindow
property theStatusPanel : null
property foundMessages : {}
property mailboxesLoaded : false
```

The value for the `theWindow` property, `forWindow`, is passed to the `makeController` handler.

The controller script initialization handler sets up columns in the data handlers that provide data for the mailboxes and messages views in the search window. This handler is shown in Listing 9-11.

> **Note:** You'll see the character ¬ (created by typing Option-l) in many of the listings in this chapter. That's the AppleScript continuation character. When a line ends with a continuation character, the next line is considered to be part of that line. To accommodate long script statements, listings in this tutorial use continuation characters that do not appear in Mail Search's script file, and in fact, you should not need to use them—see "How Xcode Formats Scripts" (page 69).

Listing 9-11 The controller script's `initialize` handler

```
on initialize()
    -- Add a column to the mailboxes data source
    tell scroll view "mailboxes" of split view 1 of theWindow
        make new data column at the end of the data columns of data source ¬
            of outline view "mailboxes" with properties {name:"mailboxes"}
    end tell

    -- Add the columns to the messages data source
    tell scroll view "messages" of split view 1 of theWindow
        make new data column at the end of the data columns of data source ¬
            of table view "messages" with properties {name:"from"}
        make new data column at the end of the data columns of data source ¬
            of table view "messages" with properties {name:"subject"}
        make new data column at the end of the data columns of data source ¬
            of table view "messages" with properties {name:"mailbox"}
    end tell

    set windowCount to windowCount + 1
end initialize
```


2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

The `initialize` handler performs the following steps:

1. It adds a mailboxes column with the name “mailboxes” to the data source of the Mailboxes outline view. The data source supplies the outline view with the data to display (account and mailbox names).
2. Similarly, it adds columns for “from”, “subject”, and “mailbox” to the data source of the Messages table view. The data source supplies the table view with the data to display (from names, subject lines, and mailbox names).

Finding and Displaying Accounts and Mailboxes

This section describes the handlers in the controller script Mail Search uses to find and display mailboxes in the search window. The jumping off point for displaying mailboxes is the `loadMailboxes` handler, shown in Listing 9-12. It is called from the `became main` handler, shown in Listing 9-5 (page 190), to ensure that whenever a window is activated it displays the available mailboxes. The `loadMailboxes` handler is responsible for loading all available mailboxes from all available accounts, if they have not already been loaded, and for showing the status dialog while loading.

The logic for the process of loading mailboxes is as follows:

1. `loadMailboxes` kicks off the process by calling `addMailBoxes`.
2. Mailboxes can reside in any account, so for each account, `addMailBoxes` calls `addAccount`.
3. For each mailbox the account contains, `addAccount` calls `addMailbox`.
4. `addAccount` adds the mailbox to the outline view of mailboxes available to search.

Listing 9-12 The controller script’s `loadMailboxes` handler

```
on loadMailboxes()
    if not mailboxesLoaded then
        -- Open the status panel
        set theStatusPanel to makeStatusPanel(theWindow)
        tell theStatusPanel to openPanel("Looking for Mailboxes...")

        -- Add the mailboxes
        addMailboxes()

        -- Close the status panel
        tell theStatusPanel to closePanel()

        set mailboxesLoaded to true
    end if
end loadMailboxes
```

The `loadMailboxes` handler performs the following steps:

1. If the mailboxes have already been loaded, it does nothing. Otherwise it performs the following steps.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

2. It calls the `makeStatusPanel` handler to create a status dialog script object, and stores a reference to it in the `theStatusPanel` property. The `makeStatusPanel` handler is described in "Write Handlers for Working With the Status Dialog" (page 204).

3. It calls the `openPanel` handler of the status dialog script object to start displaying the dialog, with the message "Looking for Mailboxes..." The `openPanel` handler is described in "The Status Dialog Script" (page 131).

4. It calls the controller handler `addMailboxes` to get all available mailboxes in all available accounts.

5. On completion of the previous step, it closes the status dialog.

6. It sets the controller property `mailboxesLoaded` to `true` (so it won't load the mailboxes if they've already been loaded).

The `addMailboxes` handler is shown in Listing 9-13. It is called from the `loadMailboxes` handler, shown in Listing 9-12. The `addMailboxes` handler is responsible for iterating over all available accounts to obtain their mailboxes.

Listing 9-13 The controller script's `addMailboxes` handler

```
on addMailboxes()
    tell application "Mail"
        set accountIndex to 0
        repeat with a in (get accounts)
            try
                set accountIndex to accountIndex + 1
                my addAccount(a, accountIndex, account name of a)
            end try
        end repeat
    end tell
end addMailboxes
```

The `addMailboxes` handler performs the following steps:

1. It targets the Mail application.

2. It gets a list of all accounts from the application.

3. It uses a repeat statement to iterate over the accounts.

4. For each account, it calls the controller handler `addAccount` (shown in Listing 9-14), passing among other things the name of the account.

   Within the `tell application "Mail"` statement block, the `addMailboxes` handler uses the term `my addAccount` to specify that it is calling another handler in the controller script.

The `addAccount` handler (shown in Listing 9-14) is called from the `addMailboxes` handler (shown in Listing 9-13). The `addAccount` handler is responsible for getting all available mailboxes in the passed account and adding them to the data source for the mailboxes view in the search window.

Listing 9-14 The controller script's `addAccounts` handler

```
on addAccount(a, accountIndex, accountName)
    -- Add a new item
```


2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

```
    set accountItem to make new data item at the end of the data items ¬
        of data source of outline view "mailboxes" ¬
        of scroll view "mailboxes" of split view 1 of theWindow
    set name of data cell 1 of accountItem to "mailboxes"
    set contents of data cell 1 of accountItem to accountName
    set associated object of accountItem to accountIndex

    -- Add the mail boxes
    tell application "Mail"
        set mailboxIndex to 0
        repeat with m in (get mailboxes of a)
            try
                set mailboxIndex to mailboxIndex + 1
                my addMailbox(accountItem, accountName, mailboxIndex, ¬
                    mailbox name of m)
            end try
        end repeat
    end tell
end addAccount
```

The `addAccount` handler performs the following steps:

1. It adds an account item to the data source for the mailboxes view in the search window. It also sets various information for the item, including its name (the name of the account).

2. It targets the Mail application.

3. It gets a list of all mailboxes in the account from the application.

4. It uses a repeat statement to iterate over the mailboxes.

5. For each mailbox, it calls the controller handler `addMailbox` (shown in Listing 9-15), passing among other things the name of the mailbox, to add the mailbox to the data source for the mailboxes view in the search window.

The `addMailbox` handler is called from the `addAccount` handler, shown in Listing 9-14, to add a single mailbox to the data source for the mailboxes view in the search window. The `addMailbox` handler is shown in Listing 9-15.

Listing 9-15 The controller script's `addMailbox` handler

```
on addMailbox(accountItem, accountName, mailboxIndex, mailboxName)
    -- Add a new item
    set mailboxItem to make new data item at the end of the data items ¬
        of accountItem
    set name of data cell 1 of mailboxItem to "mailboxes"
    set contents of data cell 1 of mailboxItem to mailboxName
    set associated object of mailboxItem to mailboxIndex
end addMailbox
```

The `addMailboxes` handler performs the following step:

1. It adds a mailbox item to the data source for the mailboxes view in the search window. It also sets various information for the item, including its name (the name of the mailbox).

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Finding and Displaying Messages

This section describes the handlers in the controller script Mail Search uses to find and display messages in the search window. Only messages in the specified mailboxes that contain the specified text in the specified part of the message are displayed.

The jumping off point for finding messages is the `find` handler. It is called from the `clicked` handler for the find button, shown in Listing 9-8 (page 191) and the `action` handler for the search text field, shown in Listing 9-7 (page 191). The `find` handler is responsible for gathering the search criteria, searching the selected mailboxes for matching messages, and displaying any such messages that are found. It also displays various status messages during the search. The `find` handler is shown in Listing 9-16

Listing 9-16 The controller script's `find` handler

```
on find()
    -- Get what and where to find
    set whatToFind to contents of text field "what" of theWindow
    set whereToFind to title of current menu item of popup button "where" ¬
        of theWindow

    -- Make sure that we have something to find
    if (count of whatToFind) is greater than 0 then
        -- Clear any previously found messages
        clearMessages()

        -- Setup a status panel
        set theStatusPanel to makeStatusPanel(theWindow)
        tell theStatusPanel to ¬
            openPanel("Determining the number of messages...")

        try
            -- Determine the mailboxes to search
            set mailboxesToSearch to selectedMailboxes()

            -- Determine the total number of messages to search
            set totalCount of theStatusPanel to ¬
                countMessages(mailboxesToSearch)

            -- Adjust the status panel
            tell theStatusPanel to adjustPanel()

            -- Find the messages
            set foundMessages to findMessages(mailboxesToSearch, ¬
                whereToFind, whatToFind)

            -- Change the status panel
            tell theStatusPanel to changePanel("Adding found messages...")

            -- Add the found messages to the result table
            addMessages(foundMessages)

            -- Close the status panel
            tell theStatusPanel to closePanel()
        on error errorText
            tell theStatusPanel to closePanel()
            display alert "AppleScript Error" as critical ¬
```


2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

```
                attached to theWindow message errorText
        end try
    else
        display alert "Missing Value" as critical attached to theWindow ¬
            message "You need to enter a value to search for."
    end if
end find
```

The `find` handler performs the following steps:

1. It gets the search criteria: the contents of the search text field (the "what" field) and the title of the location pop-up (the "where" menu).

2. If there is no search text, it displays an error message "Missing Value". Otherwise it performs the following steps.

3. It calls the controller handler `clearMessages` to clear any previous found messages in the Messages table view.

4. It calls the `makeStatusPanel` handler to create a status dialog script object, and stores a reference to it in the `theStatusPanel` property. The `makeStatusPanel` handler is described in "Write Handlers for Working With the Status Dialog" (page 204).

5. It calls the `openPanel` handler of the status dialog script object to start displaying the panel, with the message "Determining the number of messages..." The `openPanel` handler is described in "The Status Dialog Script" (page 131).

6. It sets up an error handler (a `try`…`on error`…`end try` statement) around the statements that search for and display messages. If an error occurs, the `on error` clause closes the status dialog and displays an error message. Within the handler, it performs these steps:

    a. It calls the controller handler `selectedMailboxes` to get the selected mailboxes.

    b. It calls the controller handler `countMessages` to set the total count property of the status dialog to the total number of messages to search.

    c. It calls the `adjustPanel` handler of the status dialog script object to display the number of the messages to be searched. The `adjustPanel` handler is described in "The Status Dialog Script" (page 131).

    d. It calls the controller handler `findMessages` (shown in Listing 9-17), passing the mailboxes to search, the search location, and the text to find.

    e. It calls the `changePanel` handler of the status dialog script object to display the message "Adding found messages..." The `changePanel` handler is described in "The Status Dialog Script" (page 131).

    f. It calls the controller handler `addMessages`, to display the found messages in the search window's message view.

    g. It closes the status dialog.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

The `findMessages` handler is shown in Listing 9-17. It is called from the `find` handler, shown in Listing 9-16. The `findMessages` handler is responsible for iterating over the specified mailboxes and finding any messages that contain the specified text in the specified location (From, To, or Contents of the message).

Listing 9-17 The controller script's `findMessages` handler

```
on findMessages(mailboxesToSearch, whereToFind, whatToFind)
    -- Initialize the result
    set messagesFound to {}

    tell application "Mail"
        -- Search through each of the mail boxes
        repeat with b in (get mailboxesToSearch)
            try
                -- Search through each of the messages of the mail box
                repeat with m in (get messages of b)
                    try
                        if whereToFind is equal to "Subject" then
                            if whatToFind is in the subject of m then
                                copy m to end of messagesFound
                            end if
                        else if whereToFind is equal to "From" then
                            if whatToFind is in sender of m then
                                copy m to end of messagesFound
                            end if
                        else if whereToFind is equal to "To" then
                            set foundRecipient to false

                            -- Recipients
                            repeat with r in (get recipients of m)
                                if whatToFind is in address of r or whatToFind ¬
                                        is in display name of r then
                                    set foundRecipient to true
                                end if
                            end repeat

                            -- To Recipients
                            if not foundRecipient then
                                repeat with r in (get to recipients of m)
                                    if whatToFind is in address of r or whatToFind ¬
                                            is in display name of r then
                                        set foundRecipient to true
                                    end if
                                end repeat
                            end if

                            -- cc Recipients
                            if not foundRicipient then
                                repeat with r in (get cc recipients of m)
                                    if whatToFind is in address of r or whatToFind ¬
                                            is in display name of r then
                                        set foundRecipient to true
                                    end if
                                end repeat
                            end if

                            -- bcc Recipients
```


2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

```
                    if not foundRicipient then
                        repeat with r in (get bcc recipients of m)
                            if whatToFind is in address of r or whatToFind ¬
                                    is in display name of r then
                                set foundRecipient to true
                            end if
                        end repeat
                    end if

                    if foundRecipient then
                        copy m to end of messagesFound
                    end if
                else if whereToFind is equal to "Contents" then
                    if whatToFind is in the content of m then
                        copy m to end of messagesFound
                    end if
                end if

                -- Update the status panel
                tell theStatusPanel to incrementPanel()
            end try
        end repeat
    end try
  end repeat
end tell

-- Return the result
return messagesFound
end findMessages
```

The `findMessages` handler is similar in many ways to the `find` handler. It performs the following steps:

1. It gets the location criteria: the title of the current choice of the search location pop-up (the "where" menu).

2. It targets the Mail application.

3. It sets up a Repeat loop for each selected mailbox. The repeat loop contains a `try`…`end try` error handler so that if any error occurs, the script doesn't halt. Any error is displayed by the error handler in the calling routine (`find`).

4. It sets up a Repeat loop for each message in the mailbox. This repeat loop also contains an error handler.

5. Within the inner repeat loop it does the following:

   a. It calls on the Mail application to search for the specified text in the specified location.

   b. If a message contains the specified text, it adds it to a list of found messages.

   c. At the end of the loop, it calls the `incrementPanel` handler of the status dialog script object to increment the count of the messages to be searched. The `incrementPanel` handler is described in "Write Handlers for Working With Message Windows" (page 204).

6. It returns the list of found messages. The list may be empty.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

The `find` handler (shown in Listing 9-16 (page 197)) also calls the following handlers. These handlers, which perform simple operations, aren't shown in individual listings, but you can examine them in "Chapter 10" (page 187) or in the Mail Search sample application.

- `clearMessages`: Tells the data source of the Messages table view of the controller script object's search window to delete every row, thus clearing the messages.
- `countMessages`: Communicates with the Mail application to count the messages in each mailbox in the passed list of mailboxes. Returns the total count.
- `addMessages`: Turns off updating in the Message table view. For each message in the passed list of messages, calls `addMessage`. Turns updating back on, so that the Messages table view displays the added messages.
- `addMessage`: (Not called directly by the `find` handler, but called by `addMessages`.) For the passed message, adds a row to the data source for the Messages table view, then adds a cell for each column in the row, so that the row displays the From, Subject, and Mailbox information for the message.
- `selectedMailboxes`: Gets the currently selected mailboxes from the Mailboxes outline view. If any accounts are selected, calls the `mailboxesForIndex` handler to get all corresponding mailboxes (both from accounts and from individual selected mailboxes) from the Mail application. If only mailboxes are selected, gets the mailboxes itself from the Mail Application. Returns the list of selected mailboxes (which may be empty).
- `mailboxesForIndex`: (Not called directly by the `find` handler, but called by `selectedMailboxes`.) Communicates with the Mail application to obtain the actual mailboxes corresponding to the currently selected mailboxes. Returns the list of selected mailboxes (which may be empty).

Opening Message Windows

Because the Mail application's scripting support doesn't currently allow you to open a message in a separate window, Mail Search gets the message text and displays it in its own window. This section describes the controller handlers Mail Search uses to do this.

The jumping off point for displaying messages is the `openMessages` handler. It is called from the `double clicked` handler for the Messages view in the Search window (the handler is shown in Listing 9-9 (page 192)). The Messages view currently supports only selection of single messages, so the `openMessages` handler merely calls the `openMessageWindow` handler to open the selected message. You could, however, modify the Messages table view in Interface Builder to allow multiple selection (see the Attributes pane in the Info window), then modify `openMessages` to iterate over the current selections, calling `openMessageWindow` for each selection.

The `openMessageWindow` handler is shown in Listing 9-18

Listing 9-18 The controller script's `openMessageWindow` handler

```
on openMessageWindow()
    set clickedRow to clicked row of table view "messages" ¬
        of scroll view "messages" of split view 1 of theWindow
    if clickedRow is greater than or equal to 0 then
        set theAccount to ""
        set theMailbox to ""
        set theSubject to ""
        set theDateReceived to ""
        set theContents to ""
```


2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

```
        set theSender to ""
        set theRecipients to ""
        set theCCRecipients to ""
        set theReplyTo to ""

        tell application "Mail"
            set theMessage to Abstract object clickedRow of foundMessages

            set theAccount to account name of account of container of theMessage
            set theMailbox to mailbox name of container of theMessage
            set theSubject to subject of theMessage
            -- set theDateReceived to date received of theMessage
            set theContents to content of theMessage
            set theSender to sender of theMessage
            set theRecipients to address of every recipient of theMessage
            set theCCRecipients to address of every cc recipient of theMessage
            set theReplyTo to reply to of theMessage
        end tell

        set messageWindow to makeMessageWindow()
        tell messageWindow
            set messageContents to "Account: " & theAccount & return
            set messageContents to messageContents & "Mailbox: " & theMailbox & return
            if length of theSender > 0 then
                set messageContents to messageContents & "From: " & theSender & return
            end if
            if length of theDateReceived as string > 0 then
                set messageContents to messageContents & "Date: " ¬
                    & (theDateReceived as string) & return
            end if
            if length of theRecipients > 0 then
                set messageContents to messageContents & "To: " ¬
                    & theRecipients & return
            end if
            if length of theCCRecipients > 0 then
                set messageContents to messageContents & "Cc: " ¬
                    & theCCRecipients & return
            end if
            if length of theSubject > 0 then
                set messageContents to messageContents & "Subject: " ¬
                    & theSubject & return
            end if
            if length of theReplyTo > 0 then
                set messageContents to messageContents & "Reply-To: " ¬
                    & theReplyTo & return & return
            end if
            set messageContents to messageContents & theContents
            set contents of text view "message" of scroll view "message" ¬
                to messageContents
            set title to theSubject
            set visible to true
        end tell
    end if
end openMessageWindow
```

The `openMessageWindow` handler performs the following steps:

1. It gets the row number for the currently selected message from the Messages table view.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

2. If there is no currently selected row (the row number is less than zero), it does nothing. Otherwise it performs the following steps.

3. It initializes some local variables to store message information, such as the account, mailbox, subject, and so on.

4. It calls on the Mail application to obtain an object representing the message at the selected row. The term `Abstract object` is a Cocoa scripting term specifying an object that is the parent for all script objects that have no other parent class.

5. Continuing to use the Mail application, it sets local variables to message information from the message object returned in the previous step.

6. It calls the `makeMessageWindow` handler to create a new message window for displaying the message. The `makeMessageWindow` handler is described in "Write Handlers for Working With Message Windows" (page 204).

7. It sets another local variable to the message contents of the found message by concatenating the message information previously stored in local variables, then sets the text contents of the new message window.

8. It sets the title of the new window to the subject of the found message.

9. It sets the visible property of the new window to display it to the user.

Write Handlers for Working With Controllers

Mail Search needs several handlers for working with controllers. These handlers are not part of the controller script itself. They aren't shown in individual listings, but you can examine them in "Chapter 10" (page 187) or in the Mail Search sample application.

- `makeController`: This handler contains the entire controller script. You've examined most of the handlers in this script in previous sections. The `makeController` handler is called by the `will open` handler, shown in Listing 9-4 (page 190), to create a controller script object for a search window. Since the script is the only content of the `makeController` handler, calling the handler effectively returns the script object as a return result.

- `addController`: This handler is also called by the `will open` handler, shown in Listing 9-4 (page 190). It simply adds a controller to the global list of controllers.

- `removeController`: This handler is also called by the `will close` handler, shown in Listing 9-6 (page 190). It simply removes a controller from the global list of controllers by calling the utility handler `deleteItemInList`.

- `controllerForWindow`: This handler is called whenever a handler needs to obtain the controller for the current window. It returns the controller for the passed window from the global list of controllers.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Write Handlers for Working With the Status Dialog

Mail Search only needs one handler to work with the status dialog script, the `makeStatusPanel` handler, which creates the status dialog script object. This handler is shown in full in "Chapter 10" (page 187). The handlers and properties it contains are described in "The Status Dialog Script" (page 131). Since the script is the only content of the `makeStatusPanel` handler, calling the handler effectively returns the script object as a return result.

Write Handlers for Working With Message Windows

Mail Search only needs one handler to work with the message window, the `makeMessageWindow` handler. This handler, shown in Listing 9-19, is called from the `openMessageWindow` handler (shown in Listing 9-18.

Listing 9-19 The `makeMessageWindow` handler

```
on makeMessageWindow()
    load nib "Message"
    set windowCount to windowCount + 1
    set windowName to "message " & windowCount
    set name of window "message" to windowName
    return window windowName
end makeMessageWindow
```

The `makeMessageWindow` performs the following steps:

1. It loads an instance of the Message window from the Message nib.

2. It increments Mail Search's global window count property.

3. It constructs a name for the window.

4. It returns the window name.

Write Utility Handlers

Mail Search needs one utility handler, `deleteItemInList`, to remove an item from a list. This handler is called by `removeController`, which in turn is called by the window's `will close` handler to remove a controller before the window closes. AppleScript doesn't currently support deleting items from lists directly, so scripters rely on a utility handler such as `deleteItemInList`, shown in Listing 9-20.

Listing 9-20 Utility function to delete an item from a list

```
on deleteItemInList(x, theList)
    (* To Be Provided *)
end deleteItemInList
```


2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Mail Search Tutorial: Build and Test the Application

In this chapter you'll build and test Mail Search, an AppleScript Studio application that searches for specified text in messages in the Mac OS X Mail application.

This chapter assumes you have completed previous Mail Search tutorial chapters. By now, you should have completed constructing the application, including building the interface, connecting objects in the interface to handlers in the application, and writing the handlers. Chances are you built the application several times while creating the interface, and perhaps later as well.

You may have inserted handlers in Mail Search's script file (`Mail Search.applescript`) as you worked on the tutorial, or you may have chosen to copy in the whole body of the script, either from "Appendix B" (page 223) or by copying it (as recommended) from the Mail Search sample application distributed with AppleScript Studio. You should now be ready to build the application and run it. These steps are described in the following sections:

- "Build and Run Mail Search" (page 205)
- "Check for Syntax Errors" (page 206)

Build and Run Mail Search

To build and run the application, you follow the same steps described earlier in this tutorial:

1. Open your Mail Search project in Xcode.
2. Type Command-R, choose Build and Run from the Build menu, or click the Build and Run button.

Xcode has a number of menu commands, keystroke equivalents, and buttons you can use to perform different kinds of build operations. You can display the help tag for any of the three build-related buttons visible in Figure 10-1 by positioning the cursor over them. In addition, each of these three buttons can operate as a pull-down menu.

- the hammer: Builds the active target. Same as Command-B, or Build in the Build menu. Its pull-down menu contains Clean and Clean All, used to clean the current target or the entire project of previously built products and files.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

- the hammer and green button: Builds the active target and runs the executable product. Same as Command-R, or Build and Run in the Build menu. Other options are available in its pull-down menu, but Build and Run is the most important function.
- the stop sign: Stops the current activity for a product. Same as Command-Option-R, or Stop in the Debug menu. The pull-down menu lists all the currently active products, allowing you to select the one to stop.

Note: The other three standard toolbar buttons are not build-related. See Xcode Help for more information on these and other build options.

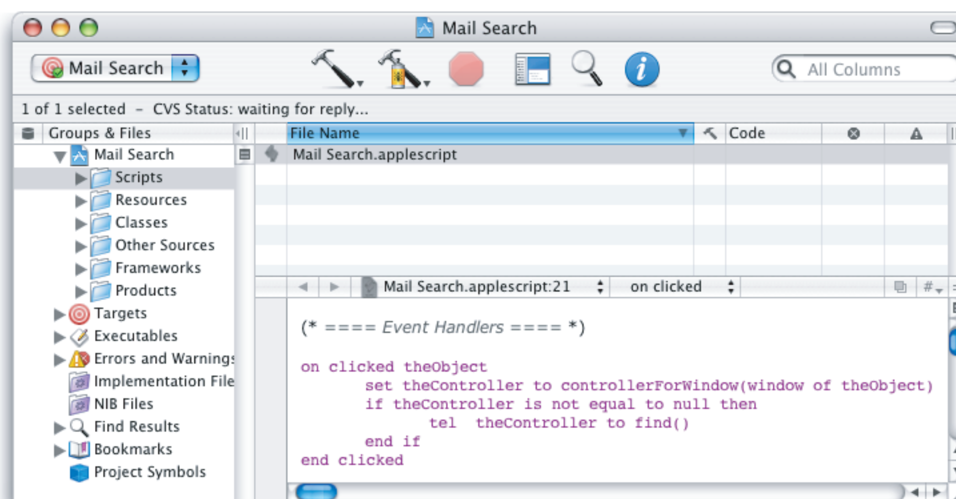
All uncompiled script files in the current target (in this case, there's just one target and one script file, `Mail Search.applescript`) are compiled automatically when you build the application. Building the application also compiles the Cocoa code in the application's `main.m` file (visible in Figure 1-10 (page 29)), prepares the application's resources, and links any designated frameworks (for Mail Search, that's the frameworks in the Linked Frameworks group within the Frameworks group in the Groups & Files list).

If the build succeeds, Mail Search opens, launches the Mail application if it isn't already running, and loads all available mailboxes. In the next sections, you'll learn what to do if Mail Search doesn't build correctly, or if testing shows the application isn't working properly.

Check for Syntax Errors

When you first enter text in an Xcode script editor window, it appears in the style specified in the Script Editor application (located in `/Applications/AppleScript`) for new (uncompiled) text. For example, Figure 10-1 shows how the `clicked` handler might look when you first type or copy it into the script file (in this case, with a minor spelling error you'll fix in a minute).

Figure 10-1 An uncompiled handler



2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

If you want to change the default format settings, use the steps described in "How Xcode Formats Scripts" (page 69). You'll have to quit and restart Xcode to pick up the new settings.

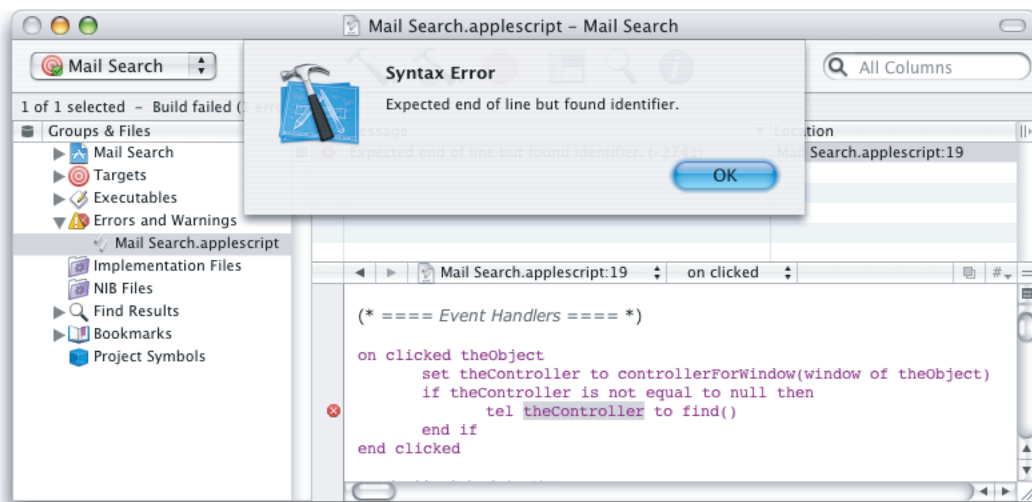
To check for syntax errors, perform these steps:

1. Open your Mail Search project in Xcode.

2. Open the Scripts group in the Files list in the Groups & Files list and select `Mail Search.applescript`. Xcode knows that files that end in `.applescript` are script files, and displays the file in a script editor window.

3. Select Build > Compile or press Command-K. Assuming your script file has the same spelling error shown in Figure 10-1, you should see a result similar to Figure 10-2.

Figure 10-2 A syntax error in an Xcode script editor window



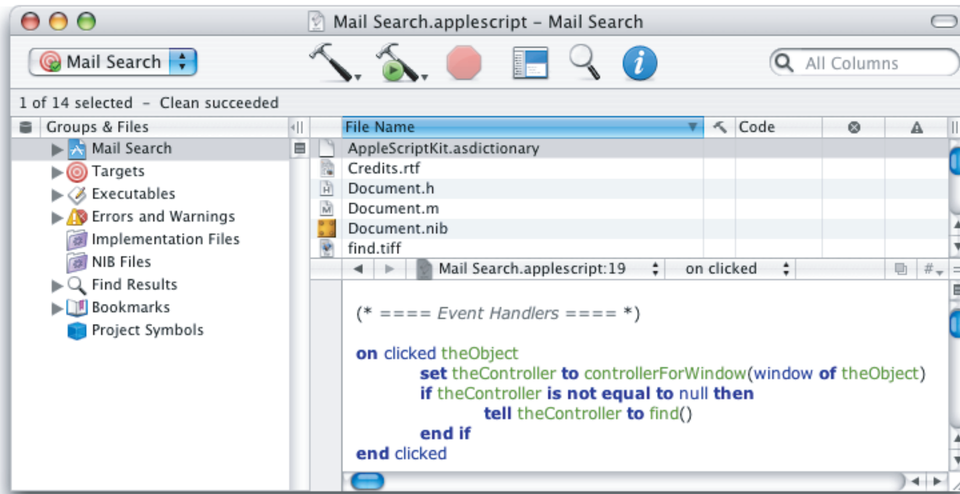
Because "tel" is misspelled (and looks like an identifier), it isn't recognized as a keyword. As a result, the next term, "theController", is flagged as an error. That word is selected and an error message is displayed, as shown in Figure 10-2.

Your job is to examine the offending line, correct whatever is wrong, and again click the checkmark to compile it. You can also initiate a compile by pressing the Enter key, typing Command-K, or choosing Compile from the Build menu. (As mentioned previously, uncompiled script files in the current target are also compiled when you build the application.) When you successfully compile the script, you should see formatting similar to that shown in Figure 10-3.

To help determine the correct terminology for a script, see "Finding Terminology Information" (page 77).

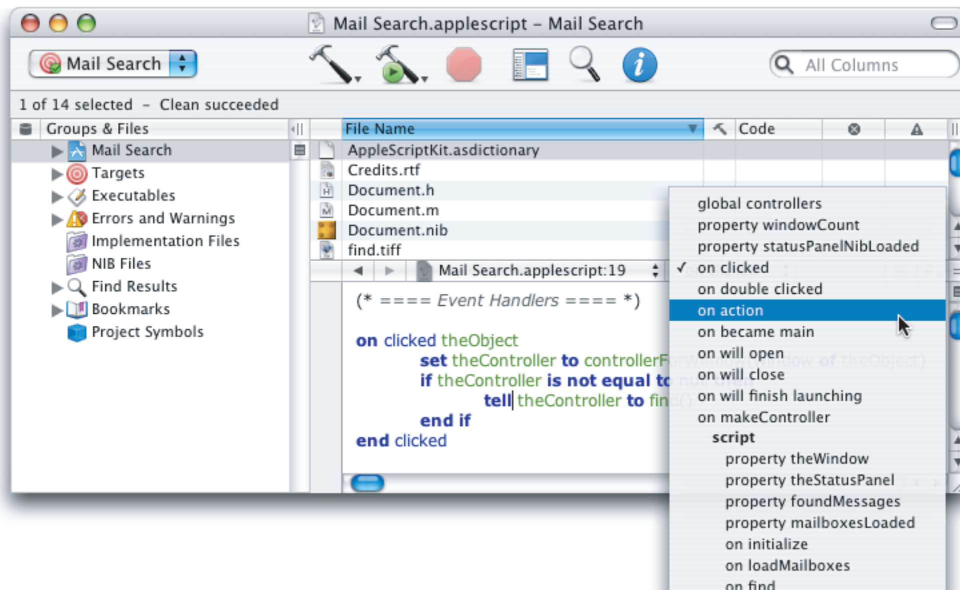
2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Figure 10-3 A compiled handler



To help in editing and compiling scripts, you can use the pop-up menu in the editor window to jump to any property, script, or handler in the script file. Figure 10-4 shows how to select the `action` handler using the pop-up menu. Once selected, the editor will jump to that handler.

Figure 10-4 Event handlers in an Xcode pop-up menu



2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Mail Search Tutorial: Customize the Application

In this chapter you'll customize Mail Search, an AppleScript Studio application that searches for specified text in messages in the Mac OS X Mail application. You'll learn how to perform basic customization that should be useful for many applications.

To customize the Mail Search application, you'll perform the steps described in the following sections:

"Customize Menus" (page 209)
"Customize the About Window" (page 212)
"Customize Version and Copyright Information" (page 214)
"Customize Icons" (page 215)

This chapter assumes you have completed previous Mail Search tutorial chapters.

Customize Menus

Figure 11-1 shows the MainMenu instance from Mail Search's `MainMenu.nib` file as it appears in Interface Builder. The application menu is open, showing the default items in that menu. You'll need to change the names of certain menu items so that they match the ones specified in "Design the Interface" (page 123).

To customize Mail Search's menus, perform the steps described in the following sections:

1. Rename the Application menu to Mail Search and add Mail Search to several of its items.
2. Optionally set menu attributes, including key equivalents.
3. Optionally remove menus and menu items.

Rename Menus and Menu Items

To rename Mail Search menus and menu items, perform the following steps:

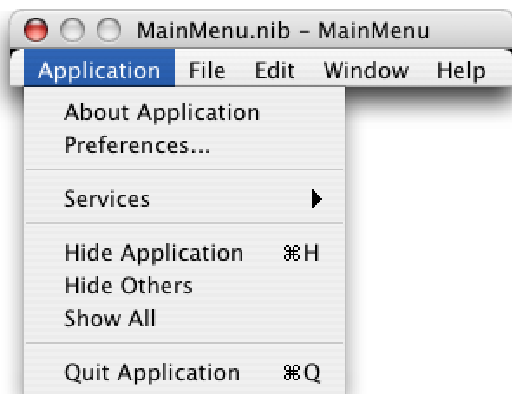
1. Open the project in Xcode.
2. Open the Resources group in the Files list in the Groups & Files list.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

3. Double-click the icon for the file `MainMenu.nib` to open the file in Interface Builder. That should automatically open the MainMenu instance, but if not, double-click it in the Instances pane.

4. Click the Application menu in the menu bar. The result should look similar to Figure 11-1.

Figure 11-1 Mail Search's menu nib in Interface Builder, showing the application menu

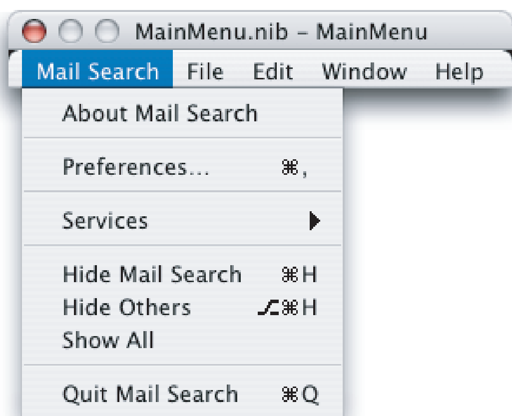


5. Double-click the About Application menu item and replace Application with Mail Search.

6. Do the same for the Hide Application and Quit Application menu items. You can tab between menu items.

7. Double-click the Application item in the menu bar and change it to Mail Search. If you open the menu again, it should now look similar to Figure 11-2

Figure 11-2 The revised Mail Search application menu



2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Set Menu Attributes

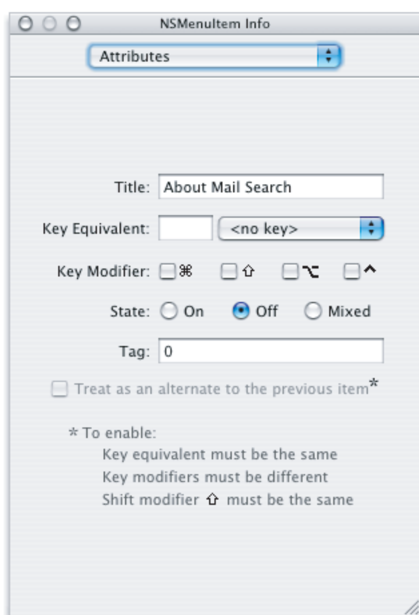
Mail Search doesn't currently supply any menu items beyond the default items that come with any AppleScript Studio document-based project. However, you may choose to modify Mail Search's menus by adding keystroke equivalents (including modifier keys) and setting the initial checked state. To do so, perform the following steps:

1. Open the `MainMenu.nib` file as described in "Rename Menus and Menu Items" (page 209).

2. Click a menu in the menu bar (in this case, choose the Mail Search menu). The result is shown in Figure 11-2.

3. Click to select the About Mail Search menu item, then type Command-Shift-I to open the Info window. If necessary, use the pop-up menu to choose the Attributes pane. The result is shown in Figure 11-3

Figure 11-3 The Info window for the About Mail Search menu item



4. To specify a key equivalent, you can type any key not used by another Mail Search menu in the Key Equivalent text field. Many keys have standard usages, so avoid overriding those keys.

5. To assign a modifier key or combination, you click to select any of the checkboxes in the Modifiers section. The checkbox on the left represents the Shift key, the middle checkbox the Option key, and the last checkbox the Control key.

6. You use the radio buttons in the State section to specify whether a menu item is initially checked.

As you can see in Figure 11-3, you can also change a menu item's name in the Attributes pane by typing in the Title field.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Remove Menus and Menu Items

You can modify Mail Search's menus by removing menus or menu items. To do so, perform the following steps:

1. Open the `MainMenu.nib` file in Interface Builder as described in "Rename Menus and Menu Items" (page 209)
2. Click the any menu in the menu bar. To delete that whole menu, simply press the Delete key or choose Delete from the Edit menu.
3. To delete a single item from a menu, open the menu as in previous sections, select the item, and press the Delete key.

Don't delete items from the Edit menu—items such as Cut and Paste are automatically supported when users open a message window in Mail Search.

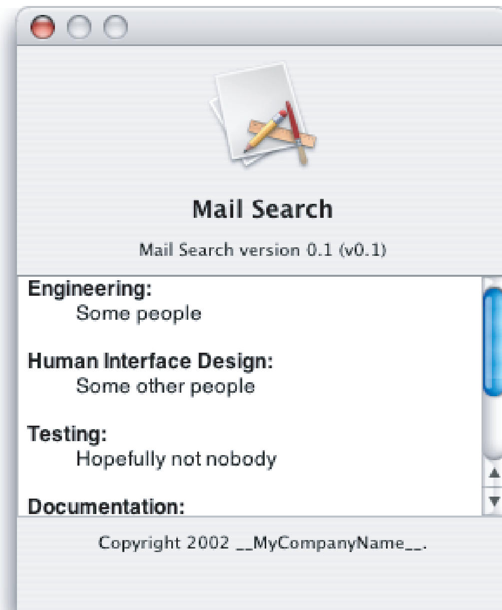
Customize the About Window

Customizing Mail Search's About window requires several simple tasks. A document-based AppleScript Studio project contains a file named `Credits.rtf`, described in "Document-based Application Template" (page 46). This rich text format file supplies the application description for the default About window. The current default About window is shown in Figure 11-4.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Note: For a simple (non-document) AppleScript application project, by default the About box displays information from the `InfoPlist.strings` file, located in the project's Resources group.

Figure 11-4 AppleScript Studio's default About window



To customize this text, you simply edit the file and supply your own information. You can open the file in Xcode, or in another editor such as TextEdit (distributed with Mac OS X). Figure 11-5 shows the About window after modifying the `Credits.rtf` file.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Figure 11-5 The About window after modifying the application description

You'll make changes in later sections that will further customize the About window:

- When you add customized information in "Customize Version and Copyright Information" (page 214), the application name, version string, and copyright information are displayed in the About window.
- When you add a custom icon in "Customize Icons" (page 215), the icon is displayed in the About window.

Customize Version and Copyright Information

An **information property list** is a special property list that contains predefined keys for application information that may be used by the Finder, by other applications, and by the application itself. The file `InfoPlist.strings` is a property list that contains application information that can be displayed to the user in several places, including:

- in Mail Search's About window
- an Info window in the Finder
- by other applications that look for the information, such as the Apple System Profiler application (located in `/Applications/Utilities`)

Listing 11-1 shows the default `InfoPlist.strings` file for an AppleScript Studio application. "Application" would reflect your product's name.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Listing 11-1 The default InfoPlist.strings file from an AppleScript Studio application

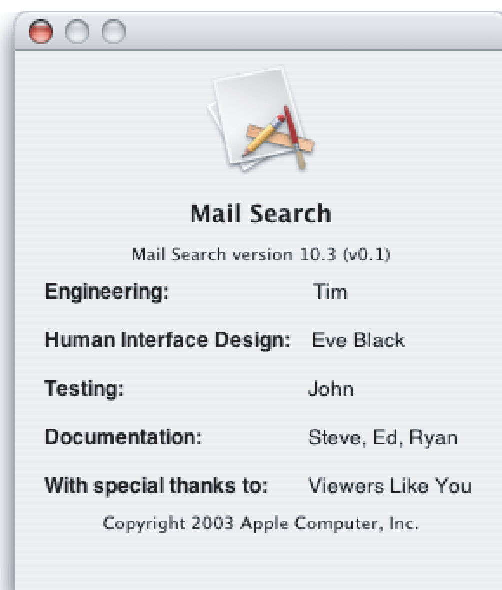
```
/* Localized versions of Info.plist keys */

CFBundleName = "Application";
CFBundleShortVersionString = "Application version 0.1";
CFBundleGetInfoString = "Application version 0.1, Copyright 2003 __MyCompanyName__.";
NSHumanReadableCopyright = "Copyright 2003 __MyCompanyName__.";
```

The string associated with the `CFBundleName` key appears as the application name in the default About window shown in Figure 11-4. The string associated with the `CFBundleShortVersionString` key appears as the application version number, below the application name, and the string associated with `NSHumanReadableCopyright` appears as the application copyright, below the application description (described in "Customize the About Window" (page 212)).

To customize version and copyright information, you simply edit the `InfoPlist.strings` file in Xcode and insert the information you want to display. Figure 11-6 shows the About window after modifying the version and copyright information.

Figure 11-6 The About window after modifying version and copyright information



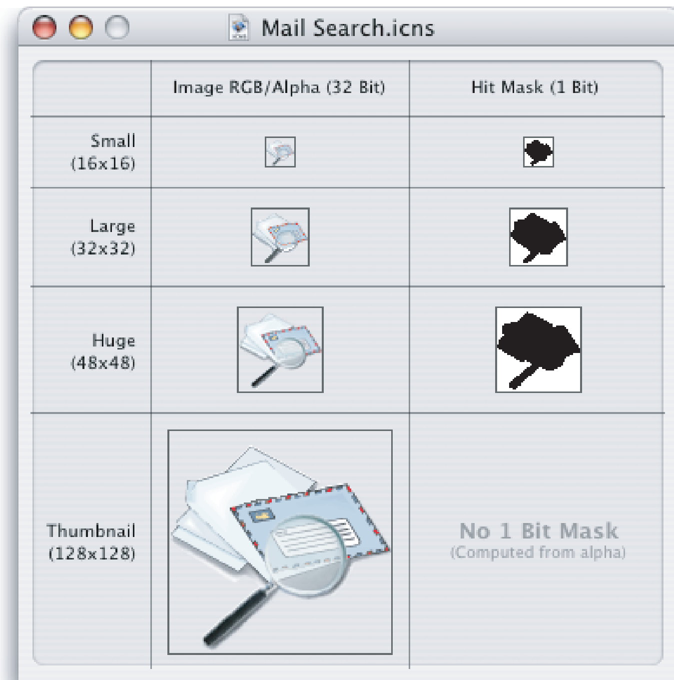
Customize Icons

Mac OS X supports the display of very large icons for the desktop, the Dock, and in various other locations. The Finder uses a high-quality scaling algorithm to generate the variable-sized icons it needs. To help ensure a pleasing result, applications should provide at least a thumbnail icon (a large, 128 x 128 image) as part of an `'icns'` resource (stored in an icon resource file with the extension ".icns").

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

The Mail Search sample application provides customized icons in the file `Mail Search.icns`. Figure 11-7 shows this file as displayed by the Icon Composer application (which is located in `/Developer/Applications/Utilities/`).

Figure 11-7 Mail Search's icons displayed in Icon Composer



To add custom icons to the Mail Search tutorial application requires the following steps:

1. Create the icon art.

2. Populate an icon resource file with the required icons (Figure 11-7 shows the contents of a fully-populated Icon Composer icon resource template).

3. Add the icon resource file to the Mail Search application.

4. Register a unique creator code for the application so that the Finder can display the correct icons for the application

The first two steps are not included here, but are described in other documentation. For example, *Learning Cocoa* by O'Reilly & Associates describes how to create a simple icon resource file with Icon Composer. For this tutorial, you can use the icon resource file provided by the Mail Search sample application.

Steps 3 and 4 are described in the following sections.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Add an Icon Resource File to the Project

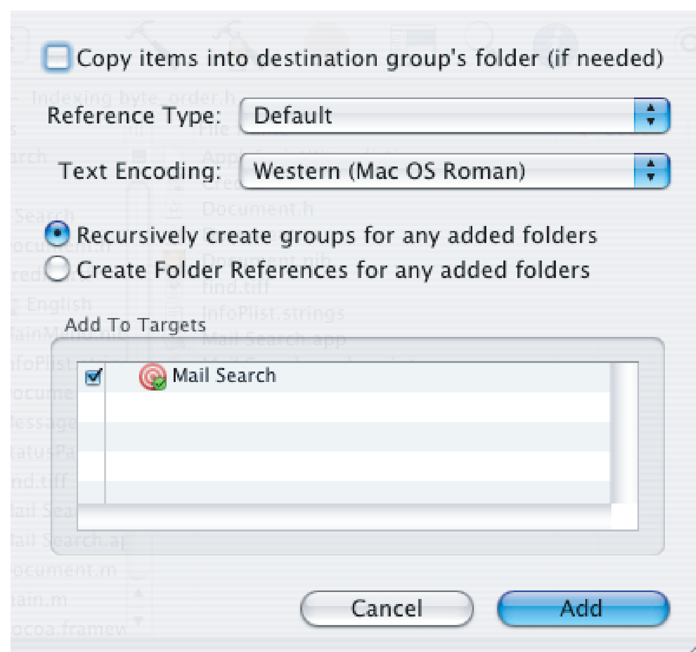
To add an icon resource file to the Mail Search tutorial project, perform these steps:

1. Open the Mail Search project in Xcode.

2. The Mail Search sample project included with AppleScript Studio includes an icon resource file that contains customized icons. Open the folder for the Mail Search sample project and drag the file `Mail Search.icns` to the Files list in the Groups & Files list of the Mail Search project. You can insert it directly in the Resources group.

   You can also add this file to the Mail Search project by choosing Add Files from the Project menu in Xcode, as described in "Create a Project" (page 133). In either case, you'll get the dialog shown in Figure 11-8.

   Figure 11-8 Adding a file to a project

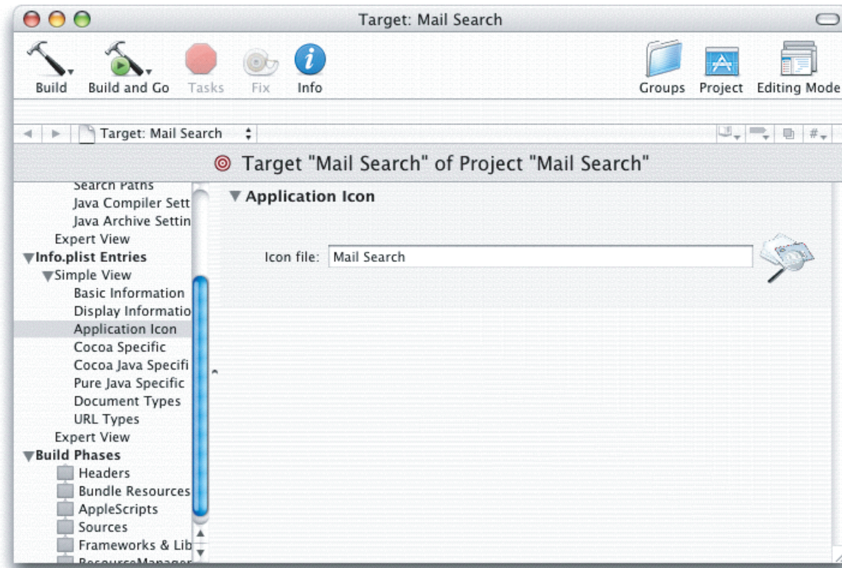
   

   Select "Copy items into destination group's folder (if needed)," then click Add to add the file to your project. If you used the Add Files menu choice, drag the file `Mail Search.icns` into the Resources group in the Groups & Files list.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

3. Double-click the Mail Search target in Xcode's Targets group to open a window for the target. Use the disclosure triangles in the column view on the left to display Simple View within the Info.plist Entries section. Then click the Application Icon entry. The result is shown in Figure 11-9.

Figure 11-9 The Icon field in a target window for the Mail Search target

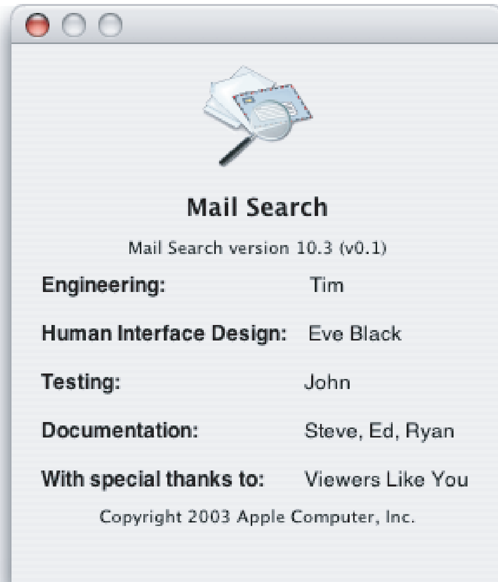


4. Type the name of the file "Mail Search" into the Icon file text field. Note you do not add the `.icns` suffix.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

5. Build and run the Mail Search application. The new About window, now including the Mail Search icon, is shown in Figure 11-10.

Figure 11-10 The About window after customizing icons

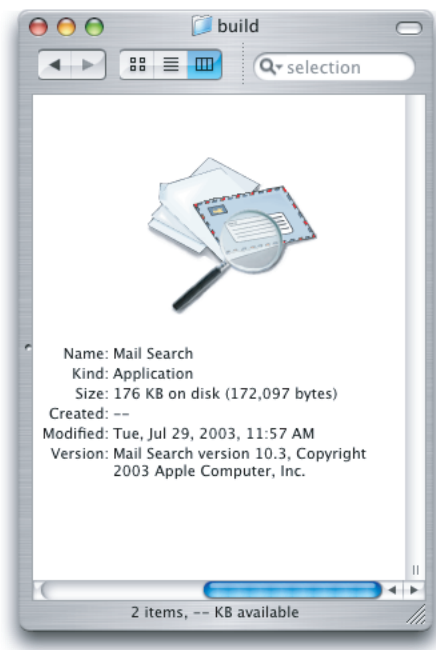


6. You may need to quit Xcode and restart the Finder to make sure the Finder recognizes the new icons.

Figure 11-11 shows the Mail Search icon in the Finder, at maximum resolution. Note the version string you added earlier shows up now in the Finder.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Figure 11-11 The Mail Search icon in the Finder



Supply a Creator Code

You should make sure your application has a unique creator code (or signature). The creator code identifies the application to the Finder so that it can display the correct icons for the application. If you create an AppleScript Studio application to distribute commercially, you should register your creator code with Apple Developer Technical Support, which keeps a database of creator codes to avoid conflict between applications.

Creator codes consisting entirely of lower case letters are reserved for Apple, so use at least one upper case letter in your code. You can make sure your creator code is unique (and also register it) at the following site:

http://developer.apple.com/datatype/

In Figure 11-9 (page 218), the Applications Settings pane shows the application type (APPL, or application) and signature (????, the default value supplied by Xcode). If you look at the same pane for the Mail Search sample project, you will see that it has the creator code “wats”.

To add your unique, four-character creator code, type it in the Signature field on the Application Settings pane.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

AppleScript Studio System Requirements and Version Information

This appendix describes the system requirements for developing and running AppleScript Studio applications and explains how to determine if AppleScript Studio is currently installed.

To build AppleScript Studio applications, you must install a version of the Mac OS X Developer Tools that includes AppleScript Studio. To run an AppleScript Studio application, the target machine must have the AppleScript Studio runtime required for the application. An AppleScript Studio runtime is available if `AppleScriptKit.framework` is present in `/System/Library/Frameworks`.

AppleScript Studio attempts to maintain the following:

- An application created and built with an older version of AppleScript Studio can run with a newer runtime.
- An application created and built with a newer version of AppleScript Studio can run with an older runtime, if it doesn't use any features introduced after that runtime.

For example, an application built with AppleScript Studio version 1.1 that uses features added in version 1.1 requires the 1.1 runtime. However, a similar application that doesn't use any features from AppleScript Studio 1.1 can run with the 1.0 runtime. And an application built with AppleScript Studio version 1.0 can run with any runtime, through version 1.3.

Important: An application created with AppleScript Studio version 1.2 does not run with any earlier runtimes, even if it does not use any new features of 1.2. This has been fixed in AppleScript Studio version 1.2.1.

Table A-1 lists AppleScript Studio versions, the development environment they are part of, and the system software the corresponding runtime is installed with.

Table A-1 Availability for AppleScript Studio development environment and runtime

AppleScript Studio Version	Distributed with development environment	Runtime installed with
1.0	December 2001 Developer Tools CD	Developer Tools CD (with AppleScript 1.8.2), or Mac OS X version 10.1.2 software update (with AppleScript 1.8.3 or later)

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

AppleScript Studio Version	Distributed with development environment	Runtime installed with
1.1	April 2002 Developer Tools CD	Developer Tools CD (with AppleScript 1.8.2 or later)
1.2	Mac OS X version 10.2 Developer Tools CD	Mac OS X version 10.2, and later (with AppleScript 1.9.0 or later)
1.2.1	December 2002 Developer Tools CD	Developer Tools CD (with AppleScript 1.9.1), or Mac OS X version 10.2.3 software update (with AppleScript 1.9.1 or later)
1.3	Mac OS X version 10.3 Xcode Tools	Mac OS X version 10.3, and later (with AppleScript 1.9.2 or later)
1.4	Mac OS X version 10.4 Xcode Tools	Mac OS X version 10.4, and later (with AppleScript 1.10 or later)

For an example of how your application can determine whether the required version of AppleScript Studio is present, see the Examples section for the `will finish launching` event handler in the Application Suite in *AppleScript Studio Terminology Reference*.

Starting with the version of Interface Builder released with Mac OS X version 10.2, there is a Nib File Compatibility preference on the General pane of the Interface Builder Preferences window. You should select a nib-file preference that suits your compatibility goals, from the following choices (and restart Interface Builder for the changes to take affect):

- **Pre-10.2 format:** applications will run with earlier versions of Mac OS X, but will not have access to new features (such as the circular progress indicator or the brushed-metal, textured window appearance)
- **10.2 and later format:** provides access to all new features, but is not guaranteed to run in earlier versions of Mac OS X
- **Both Formats:** provides access to new features but will also run in earlier versions of Mac OS X (though without the new features)

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Mail Search Tutorial, Full Script Listing

The listings in this appendix provide the full source code for the Mail Search tutorial, from the script file `Mail Search.applescript`, which you can also obtain as part of the Mail Search sample application.

Important: The scripting terminology for the Mail application changed with Mac OS X version 10.2, so there are some differences between the Mail Search script file distributed with AppleScript Studio 1.2. and the listings shown in this chapter. The Mail Search application distributed with AppleScript Studio 1.2 will work with the newer version of Mail.

After installing AppleScript Studio, the sample applications are located in `/Developer/Examples/AppleScript Studio`.

Note: In AppleScript Studio 1.0, the Mail Search sample application was known as "Watson".

Listing B-1 Mail Search's global variables and event handlers

```
(* Mail Search.applescript *)

(* ==== Globals ==== *)

global controllers

(* ==== Properties ==== *)

property windowCount : 0
property statusPanelNibLoaded : false

(* ==== Event Handlers ==== *)

on clicked theObject
    set theController to controllerForWindow(window of theObject)
    if theController is not equal to null then
        tell theController to find()
    end if
end clicked

on double clicked theObject
    set theController to controllerForWindow(window of theObject)
    if theController is not equal to null then
        tell theController to openMessages()
```


2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

```
    end if
end double clicked

on action theObject
    set theController to controllerForWindow(window of theObject)
    if theController is not equal to null then
        tell theController to find()
    end if
end action

on became main theObject
    set theController to controllerForWindow(theObject)
    if theController is not equal to null then
        tell theController to loadMailboxes()
    end if
end became main

on will open theObject
    set theController to makeController(theObject)
    if theController is not equal to null then
        addController(theController)
        tell theController to initialize()
    end if
end will open

on will close theObject
    removeController(theObject)
end will close

on will finish launching theObject
    set controllers to {}
end will finish launching
```

Listing B-2 The controller script definition

```
(* ==== Controller Handlers ==== *)

on makeController(forWindow)
    script
        property theWindow : forWindow
        property theStatusPanel : null
        property foundMessages : {}
        property mailboxesLoaded : false

        -- Handlers

        on initialize()
            -- Add a column to the mailboxes data source
            tell scroll view "mailboxes" of split view 1 of theWindow
                make new data column at the end of the data columns of data source¬
                    of outline view "mailboxes" with properties {name:"mailboxes"}
            end tell

            -- Add the columns to the messages data source
            tell scroll view "messages" of split view 1 of theWindow
                make new data column at the end of the data columns of data source¬
                    of table view "messages" with properties {name:"from"}
                make new data column at the end of the data columns of data source¬
```


2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

```
            of table view "messages" with properties {name:"subject"}
        make new data column at the end of the data columns of data source¬
            of table view "messages" with properties {name:"mailbox"}
    end tell

    set windowCount to windowCount + 1
end initialize

on loadMailboxes()
    if not mailboxesLoaded then
        -- Open the status panel
        set theStatusPanel to makeStatusPanel(theWindow)
        tell theStatusPanel to openPanel("Looking for Mailboxes...")

        -- Add the mailboxes
        addMailboxes()

        -- Close the status panel
        tell theStatusPanel to closePanel()

        set mailboxesLoaded to true
    end if
end loadMailboxes

on find()
    -- Get what and where to find
    set whatToFind to contents of text field "what" of theWindow
    set whereToFind to title of current menu item of popup button "where"¬
        of theWindow

    -- Make sure that we have something to find
    if (count of whatToFind) is greater than 0 then
        -- Clear any previously found messages
        clearMessages()

        -- Setup a status panel
        set theStatusPanel to makeStatusPanel(theWindow)
        tell theStatusPanel to openPanel("Determining the number¬
            of messages...")

        try
            -- Determine the mailboxes to search
            set mailboxesToSearch to selectedMailboxes()

            -- Determine the total number of messages to search
            set totalCount of theStatusPanel¬
                to countMessages(mailboxesToSearch)

            -- Adjust the status panel
            tell theStatusPanel to adjustPanel()

            -- Find the messages
            set foundMessages to findMessages(mailboxesToSearch,¬
                whereToFind, whatToFind)

            -- Change the status panel
            tell theStatusPanel to changePanel("Adding found messages...")
```


2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

```
            -- Add the found messages to the result table
            addMessages(foundMessages)

            -- Close the status panel
            tell theStatusPanel to closePanel()
        on error errorText
            tell theStatusPanel to closePanel()
            display alert "AppleScript Error" as critical attached¬
                to theWindow message errorText
        end try
    else
        display alert "Missing Value" as critical attached¬
            to theWindow message "You need to enter a value to search for."
    end if
end find

on addMailbox(accountItem, accountName, mailboxIndex, mailboxName)
    -- Add a new item
    set mailboxItem to make new data item at the end of the data items¬
        of accountItem
    set name of data cell 1 of mailboxItem to "mailboxes"
    set contents of data cell 1 of mailboxItem to mailboxName
    set associated object of mailboxItem to mailboxIndex
end addMailbox

on addAccount(a, accountIndex, accountName)
    -- Add a new item
    set accountItem to make new data item at the end of the data items¬
        of data source of outline view "mailboxes"¬
            of scroll view "mailboxes" of split view 1 of theWindow
    set name of data cell 1 of accountItem to "mailboxes"
    set contents of data cell 1 of accountItem to accountName
    set associated object of accountItem to accountIndex

    -- Add the mail boxes
    tell application "Mail"
        set mailboxIndex to 0
        repeat with m in (get mailboxes of a)
            try
                set mailboxIndex to mailboxIndex + 1
                my addMailbox(accountItem, accountName, mailboxIndex,¬
                    mailbox name of m)
            end try
        end repeat
    end tell
end addAccount

on addMailboxes()
    tell application "Mail"
        set accountIndex to 0
        repeat with a in (get accounts)
            try
                set accountIndex to accountIndex + 1
                my addAccount(a, accountIndex, account name of a)
            end try
        end repeat
    end tell
end addMailboxes
```


2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

```
on mailboxesForIndex(mailboxIndex)
    -- Initialize the result
    set theMailboxes to {}

    set theIndex to 0
    set theAccountIndex to 0

    -- Determine if the selected item is an account or a mailbox
    tell outline view "mailboxes" of scroll view "mailboxes"¬
            of split view 1 of theWindow
        set theItem to item for row mailboxIndex
        set theName to contents of data cell 1 of theItem
        set theIndex to associated object of theItem
        if has parent data item of theItem then
            set theAccountIndex to the associated object¬
                of the parent data item of theItem
        end if
    end tell

    tell application "Mail"
        if theAccountIndex > 0 then
            set theMailboxes to {mailbox theIndex of account theAccountIndex}
        else
            set theMailboxes to theMailboxes & every mailbox¬
                of account theIndex
        end if
    end tell

    -- Return the result
    return theMailboxes
end mailboxesForIndex

on selectedMailboxes()
    -- Initialize the result
    set mailboxesSelected to {}

    -- Get the currently selected mailboxes in the outline view
    set mailboxIndicies to selected rows of outline view "mailboxes"¬
        of scroll view "mailboxes" of split view 1 of theWindow

    -- Get the actual mailboxes from Mail
    tell application "Mail"
        if (count of mailboxIndicies) is equal to 0 then
            repeat with a in (get accounts)
                set mailboxesSelected to mailboxesSelected &¬
                    every mailbox of a
            end repeat
        else
            repeat with i in mailboxIndicies
                set mailboxesSelected to mailboxesSelected¬
                    & my mailboxesForIndex(i)
            end repeat
        end if
    end tell

    -- Return the result
    return mailboxesSelected
```


2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

```
end selectedMailboxes

on addMessage(messageFrom, messageSubject, messageMailbox)
    -- Add a new row
    set theRow to make new data row at the end of the data rows¬
        of data source of table view "messages" of scroll view "messages"¬
            of split view 1 of theWindow

    -- Add "From" cell
    set name of data cell 1 of theRow to "from"
    set contents of data cell 1 of theRow to messageFrom

    -- Add "Subject" cell
    set name of data cell 2 of theRow to "subject"
    set contents of data cell 2 of theRow to messageSubject

    -- Add "Mailbox" cell
    set name of data cell 3 of theRow to "mailbox"
    set contents of data cell 3 of theRow to messageMailbox

    -- set the associated object of theRow to m
end addMessage

on addMessages(foundMessages)
    set update views of data source of table view "messages"¬
        of scroll view "messages" of split view 1 of theWindow to false

    tell application "Mail"
        repeat with m in foundMessages
            try
                set messageMailbox to account name of account 1
                    of container of m & "/" & mailbox name of container of m¬
                my addMessage(sender of m, subject of m, messageMailbox)
            end try
        end repeat
    end tell

    set update views of data source of table view "messages"¬
        of scroll view "messages" of split view 1 of theWindow to true
end addMessages

on findMessages(mailboxesToSearch, whereToFind, whatToFind)
    -- Initialize the result
    set messagesFound to {}

    tell application "Mail"
        -- Search through each of the mail boxes
        repeat with b in (get mailboxesToSearch)
            try
                -- Search through each of the messages of the mail box
                repeat with m in (get messages of b)
                    try
                        if whereToFind is equal to "Subject" then
                            if whatToFind is in the subject of m then
                                copy m to end of messagesFound
                            end if
                        else if whereToFind is equal to "From" then
                            if whatToFind is in sender of m then
```


2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

```
                copy m to end of messagesFound
            end if
        else if whereToFind is equal to "To" then
            set foundRecipient to false

            -- Recipients
            repeat with r in (get recipients of m)
                if whatToFind is in address of r¬
                    or whatToFind is in display name of r then
                    set foundRecipient to true
                end if
            end repeat

            -- To Recipients
            if not foundRecipient then
                repeat with r in (get to recipients of m)
                    if whatToFind is in address of r¬
                            or whatToFind is in display name¬
                            of r then
                        set foundRecipient to true
                    end if
                end repeat
            end if

            -- cc Recipients
            if not foundRicipient then
                repeat with r in (get cc recipients of m)
                    if whatToFind is in address of r¬
                            or whatToFind is in display name¬
                                of r then
                        set foundRecipient to true
                    end if
                end repeat
            end if

            -- bcc Recipients
            if not foundRicipient then
                repeat with r in (get bcc recipients of m)
                    if whatToFind is in address of r¬
                    or whatToFind is in display name of r then
                        set foundRecipient to true
                    end if
                end repeat
            end if

            if foundRecipient then
                copy m to end of messagesFound
            end if
        else if whereToFind is equal to "Contents" then
            if whatToFind is in the content of m then
                copy m to end of messagesFound
            end if
        end if

        -- Update the status panel
        tell theStatusPanel to incrementPanel()
    end try
end repeat
```


2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

```
            end try
        end repeat
    end tell

    -- Return the result
    return messagesFound
end findMessages

on clearMessages()
    tell scroll view "messages" of split view 1 of theWindow
        tell data source of table view "messages" to delete every data row
    end tell
end clearMessages

on countMessages(mailboxesToSearch)
    set messageCount to 0

    tell application "Mail"
        repeat with b in (get mailboxesToSearch)
            try
                set messageCount to messageCount + (count of every message¬
                    of b)
            end try
        end repeat
    end tell

    return messageCount
end countMessages

on openMessages()
    -- Since Mail.app currently can't open a selected message then we
    --  will just open it in our own window
    openMessageWindow()
end openMessages

on openMessageWindow()
    set clickedRow to clicked row of table view "messages"¬
        of scroll view "messages" of split view 1 of theWindow
    if clickedRow is greater than or equal to 0 then
        set theAccount to ""
        set theMailbox to ""
        set theSubject to ""
        set theDateReceived to ""
        set theContents to ""
        set theSender to ""
        set theRecipients to ""
        set theCCRecipients to ""
        set theReplyTo to ""

        tell application "Mail"
            set theMessage to Abstract object clickedRow of foundMessages

            set theAccount to account name of account of container¬
                of theMessage
            set theMailbox to mailbox name of container of theMessage
            set theSubject to subject of theMessage
            -- set theDateReceived to date received of theMessage
            set theContents to content of theMessage
```


2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

```
                set theSender to sender of theMessage
                set theRecipients to address of every recipient of theMessage
                set theCCRecipients to address of every cc recipient of theMessage
                set theReplyTo to reply to of theMessage
            end tell

            set messageWindow to makeMessageWindow()
            tell messageWindow
                set messageContents to "Account: " & theAccount & return
                set messageContents to messageContents & "Mailbox: "¬
                    & theMailbox & return
                if length of theSender > 0 then
                    set messageContents to messageContents & "From: "¬
                        & theSender & return
                end if
                if length of theDateReceived as string > 0 then
                    set messageContents to messageContents & "Date: "¬
                        & (theDateReceived as string) & return
                end if
                if length of theRecipients > 0 then
                    set messageContents to messageContents & "To: "¬
                        & theRecipients & return
                end if
                if length of theCCRecipients > 0 then
                    set messageContents to messageContents & "Cc: "¬
                        & theCCRecipients & return
                end if
                if length of theSubject > 0 then
                    set messageContents to messageContents & "Subject: "¬
                        & theSubject & return
                end if
                if length of theReplyTo > 0 then
                    set messageContents to messageContents & "Reply-To: "¬
                        & theReplyTo & return & return
                end if
                set messageContents to messageContents & theContents
                set contents of text view "message" of scroll view "message"¬
                    to messageContents
                set title to theSubject
                set visible to true
            end tell
        end if
    end openMessageWindow
  end script
end makeController
```

Listing B-3 Handlers for working with controller script objects

```
on addController(theController)
    set controllers to controllers & {theController}
end addController

on removeController(forWindow)
    set theController to controllerForWindow(forWindow)
    if theController is not equal to null then
        deleteItemInList(theController, controllers)
    end if
end removeController
```


2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

```
on controllerForWindow(aWindow)
    repeat with c in controllers
        if theWindow of c is equal to aWindow then
            set theController to c
        end if
    end repeat
    return theController
end controllerForWindow
```

Listing B-4 The message window handler

```
(* ==== Message Window Handlers ==== *)

on makeMessageWindow()
    load nib "Message"
    set windowCount to windowCount + 1
    set windowName to "message " & windowCount
    set name of window "message" to windowName
    return window windowName
end makeMessageWindow
```

Listing B-5 The status dialog script definition

```
(* ==== Status Panel Handlers ==== *)

on makeStatusPanel(forWindow)
    script
        property theWindow : forWindow
        property initialized : false
        property totalCount : 0
        property currentCount : 0

        -- Handlers
        on openPanel(statusMessage)
            if initialized is false then
                if not statusPanelNibLoaded then
                    load nib "StatusPanel"
                    set statusPanelNibLoaded to true
                end if
                tell window "status"
                    set indeterminate of progress indicator "progress" to true
                    tell progress indicator "progress" to start
                    set contents of text field "statusMessage" to statusMessage
                end tell
                set initialized to true
            end if
            display panel window "status" attached to theWindow
        end openPanel

        on changePanel(statusMessage)
            tell window "status"
                set indeterminate of progress indicator "progress" to true
                tell progress indicator "progress" to start
                set contents of text field "statusMessage" to statusMessage
            end tell
        end changePanel
```


2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

```
        on adjustPanel()
            tell progress indicator "progress" of window "status"
                set indeterminate to false
                set minimum value to currentCount
                set maximum value to totalCount
                set contents to 0
            end tell
            incrementPanel()
        end adjustPanel

        on incrementPanel()
            set currentCount to currentCount + 1
            if currentCount ≤ totalCount then
                tell window "status"
                    tell progress indicator "progress" to increment by 1
                    set contents of text field "statusMessage" to "Message "¬
                        & currentCount & " of " & totalCount
                end tell
            end if
        end incrementPanel

        on closePanel()
            close panel window "status"
        end closePanel
    end script
end makeStatusPanel
```

Listing B-6 The delete items in list utility

```
(* To be provided *)
```

Mail Search Copyright Notice

(* © Copyright 2001, 2002 Apple Computer, Inc. All rights reserved. IMPORTANT: This Apple software is supplied to you by Apple Computer, Inc. (“Apple”) in consideration of your agreement to the following terms, and your use, installation, modification or redistribution of this Apple software constitutes acceptance of these terms. If you do not agree with these terms, please do not use, install, modify or redistribute this Apple software. In consideration of your agreement to abide by the following terms, and subject to these terms, Apple grants you a personal, non-exclusive license, under Apple’s copyrights in this original Apple software (the “Apple Software”), to use, reproduce, modify and redistribute the Apple Software, with or without modifications, in source and/or binary forms; provided that if you redistribute the Apple Software in its entirety and without modifications, you must retain this notice and the following text and disclaimers in all such redistributions of the Apple Software. Neither the name, trademarks, service marks or logos of Apple Computer, Inc. may be used to endorse or promote products derived from the Apple Software without specific prior written permission from Apple. Except as expressly stated in this notice, no other rights or licenses, express or implied, are granted by Apple herein, including but not limited to any patent rights that may be infringed by your derivative works or by other works in which the Apple Software may be incorporated. The Apple Software is provided by Apple on an "AS IS" basis. APPLE MAKES NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, REGARDING THE APPLE SOFTWARE OR ITS USE AND OPERATION ALONE OR IN COMBINATION WITH YOUR PRODUCTS. IN NO EVENT SHALL APPLE BE

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

LIABLE FOR ANY SPECIAL, INDIRECT, INCIDENTAL OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) ARISING IN ANY WAY OUT OF THE USE, REPRODUCTION, MODIFICATION AND/OR DISTRIBUTION OF THE APPLE SOFTWARE, HOWEVER CAUSED AND WHETHER UNDER THEORY OF CONTRACT, TORT (INCLUDING NEGLIGENCE), STRICT LIABILITY OR OTHERWISE, EVEN IF APPLE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. *)

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Glossary

'aete' resource An Apple event terminology resource; supplies scripting terminology for a Carbon application. Compare script suite.

AppKit framework Defines classes to support a graphical, event-driven user interface for applications. See also Cocoa framework.

Apple Event Manager Provides an API for sending and receiving Apple events and working with the information they contain.

AppleScript A scripting system that allows users to directly control Macintosh applications, including the Mac OS itself, by creating sets of English-like instructions, or scripts.

AppleScript component The scripting component in Mac OS X that implements the AppleScript scripting language. A scripting component provides services for compiling and executing scripts (and relies on the **Open Scripting Architecture**).

AppleScriptKit framework Supplies advanced Cocoa scripting support and other features required by AppleScript Studio.

AppleScript object A distinct object in an application or its documents that can be specified in a script.

AppleScript object class A category for AppleScript objects that share characteristics, such as properties and elements.

AppleScript script file A file with the extension ".applescript" that contains statements in the AppleScript scripting language.

AppleScript source code editor an Xcode pane for editing and compiling AppleScript script files (files with the extension ".applescript"). The source editor relies on the `osacompile` shell tool to compile scripts.

AppleScript Studio A development environment and application framework that combines features from AppleScript, Xcode, Interface Builder, and the Cocoa application framework to provide a sophisticated environment for creating AppleScript Studio applications.

AppleScript Studio application A Mac OS X application that combines AppleScript scripts and Cocoa user-interface objects.

backtrace A list of the handlers that have been invoked at any point in a script execution. Each handler is listed as a call frame.

build directory The file system directory in which built products are stored. This is usually the "build" folder in the project folder.

build phase A step of the process of building a target. Each build phase deals with one category of source or resource files (e.g. Objective-C, AppleScript, Bundle resources, shell scripts). The Jam system automatically performs all necessary build phases in reverse order of their dependency on each other.

build style A methodology of creating a product from a target in Xcode. Development and Deployment build styles use different methodologies.

call frame The information about a handler call, including its calling parameters and local variables.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Cocoa framework An object-oriented application framework, consisting of a collection of advanced object-oriented APIs. The Cocoa framework is made up of the AppKit and Foundation frameworks. Also referred to simply as **Cocoa**.

Cocoa user interface class A class that supports a user interface item. The Application Kit provides many of these classes; for example, NSButton and NSBrowser are Cocoa user interface classes provided by the Application Kit.

Cocoa user interface object An instance of a Cocoa user interface class.

command A word or phrase in a script that requests an action. For example, a script can send a `stop` command to a progress indicator object. Compare **event**.

CVS The Concurrent Version System, a source-code control system that Xcode can use to manage changes in source code over time and across multiple developers.

data source object An object supplied by AppleScript Studio that supplies data to a table view or other view with rows and columns.

deployment build style A methodology of creating a product from a target that makes the product more appropriate for distribution to users.

development build style A methodology of creating a product from a target that makes the product more appropriate for debugging and testing.

dictionary browser See **terminology browser.**

droplet A script application that launches when you drag a file or folder icon in the Finder and "drop" it on the droplet's icon. A droplet receives a list of descriptors for the folders or files dropped on it and typically performs operations on each item in the list.

event An action an object can respond to. For example, a button click is an event that may result in execution of a `clicked` handler for the button that was clicked. Compare **command**.

event handler A handler that responds to an action in an AppleScript Studio application. Compare **handler**.

executable An application that uses a project's product and can be launched in order to debug that product. For AppleScript Studio, the executable is the product.

Foundation framework Defines a layer of useful primitive object classes, including support for Unicode strings, allocation and deallocation of objects, arrays and collections, dates, ports, and more. See also **Cocoa framework**.

framework A type of bundle (or directory in the file system) that packages software with the resources that software requires, including its interface.

GDB or gdb The GNU debugger—an open-source debugger, available with Mac OS X, for debugging programs written in C, C++, and Objective-C.

handler A named series of one or more script statements that are executed by calling its name. Compare **event handler**.

information property list A special property list that contains predefined keys for application information that may be used by the Finder, by other applications, and by the application itself. See also **property list**.

Info window An Interface Builder window for setting both attributes and connections for the associated user interface object.

Interface Builder A graphical user interface editor for creating interfaces for Cocoa, Carbon, and AppleScript Studio applications.

model-view-controller (MVC) A programming paradigm in which the view is responsible for part of the application visible on screen, the model represents the application's data and algorithms, and the controller interprets user input and specifies changes to the model and the view.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

nib A resource that stores a collection of Cocoa user interface objects, such as buttons, text fields, and pop-up menus, as well as information about the relationships between those objects and project code.

nib file A file that stores one or more nibs.

Open Scripting Architecture (OSA) An API for compiling and executing scripts, and for creating scripting components.

osacompile A shell tool for compiling script files.

Palette window An Interface Builder window that provides a number of palettes (or panes), each of which contains object instances you can add to an application.

product An application or framework produced by Xcode. AppleScript Studio projects create an application as a product.

product directory The file system directory that contains a project file, the project's source code and resources, and the build directory.

Xcode An integrated development environment for Mac OS X that supports building Cocoa, Carbon, and AppleScript Studio applications (as well as bundles, frameworks, plug-ins, and tools) with C, C++, Objective-C, and Java.

project file A file created by Xcode that organizes source code, resources, and settings used to build a product.

property list A structured, textual representation of data, commonly stored in Extensible Markup Language (XML) format. Elements of a property list represent data of certain types, such as arrays, dictionaries, and strings.

Script Editor application An application distributed with the Mac OS that provides a basic environment for editing, compiling, and executing scripts.

scripting addition Code, stored in `/System/Library/SystemAdditions`, that makes additional commands or coercions available to scripts on the same computer.

script object A user-defined object, combining data (in the form of properties) and handlers, that can be used in a script.

script object definition A compound statement that can contain collections of properties, handlers, and other AppleScript script statements.

script suite The combination of at least one suite definition and one suite terminology that together define the scripting capabilities and terminology for Cocoa and AppleScript Studio applications.

SOAP (simple object access protocol) A remote procedure call protocol designed for a distributed environment, where a server may consist of a hierarchy of objects whose methods can be called over the Internet.

suite definition A property list that describes scriptable objects in terms of their attributes, relationships, and supported commands

suite terminology A property list that maps AppleScript terminology—the English-like words and phrases you can use in a script—to the class and command descriptions in a suite definition.

target A subdivision of an Xcode project that is responsible for building one product. AppleScript Studio projects usually have only one target.

terminology browser A graphical tool for displaying the scripting terminology for a scriptable application. Also known as a **dictionary browser**.

XML-RPC A simple protocol for making remote procedure requests to Internet-based servers.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Document Revision History

This table describes the changes to *AppleScript Studio Programming Guide*.

Date	Notes
2006-04-04	Corrected sample script that checks for AppleScript Studio version and added links between sections.
	Updated script in Listing 4-2 (page 86) to work with versions of AppleScript greater than 1.9.
2005-04-29	Made minor changes to first two chapters and minor corrections throughout. Changed title from "Building Applications With AppleScript Studio."
	Added descriptions of Studio sample applications that became available in Mac OS X version 10.4. See "AppleScript Studio Sample Applications" (page 35).
	Minor corrections to Figure 1-15 (page 32) and Figure 1-17 (page 34) in "Creating a Hello World Application" (page 27) and Figure 5-5 (page 94), Figure 5-6 (page 95), and Figure 5-7 (page 96) in "Build the Interface" (page 91).
2004-04-19	In "Add an Icon Resource File to the Project" (page 217), replaced the illustration in Figure 11-9 (page 218) and corrected the text in the step that refers to it.
	Replaced the illustrations in Figure 2-2 (page 47) and Figure 7-1 (page 134), which showed a document-based project containing a Classes group (which is no longer created automatically, starting with AppleScript Studio version 1.2).
	In "Overridden Scripting Additions" (page 68), added note that usage of stop, note, and caution icons is discouraged.
	In "AppleScript Studio Xcode Plug-in Template" (page 43), elaborated on information describing plug-in search locations.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Date	Notes
2003-09-16	Notes of this date are consolidated with revision notes from August 21, 2003 to reflect changes for the final shipping version of AppleScript Studio version 1.3.
	Changed illustrations to reflect current user interface for Xcode and Interface Builder, as well as Mac OS X version 10.3.
	Changed descriptions of Script Editor, where necessary, to reflect changes in new Script Editor released with Mac OS X version 10.3. For example, see "How Xcode Formats Scripts" (page 69).
	Added information on new feature added in AppleScript Studio version 1.3: "AppleScript Studio Xcode Plug-in Template" (page 43)
	Added Table 3-2 (page 67).
	Updated Table A-1 (page 221) for Mac OS X version 10.3 and AppleScript version 1.9.2.
	Removed most documentation for line-by-line debugging with breakpoints, pending completion of the feature.
	Added descriptions for sample applications that were added in AppleScript Studio 1.2.1.
2003-02-01	Updated for latest publishing format.
	Added capability to be linked from other documents.
	Minor text corrections.
	Converted Revision History appendix to standard format.
2002-06-01	Preliminary draft of revised document.
	Added descriptions for sample applications that are new in AppleScript Studio 1.2.
	Converted "Watson" to "Mail Search" to coincide with renamed sample application. Includes new screenshots for tutorial chapters and others.
	Some new, revised, and reorganized programming tips and recipes.
	Added some information about differences between AppleScript Studio versions.
	Revised "AppleScript Studio System Requirements and Version Information" (page 221) and added table with release information through AppleScript Studio 1.2.
2002-03-01	Preliminary draft of revised document, updated for AppleScript Studio 1.1.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

Date	Notes
	Added chapter "Currency Converter Tutorial" (page 89). Moved some introductory material there from Mail Search tutorial chapters.
	Added chapter "AppleScript Studio Cookbook" (page 85), as a placeholder for recipes for common tasks.
	Created separate chapter "AppleScript Studio Components" (page 39) from "Components" sections in previous "AppleScript In Detail" chapter.
	Created separate chapter "Programming With AppleScript Studio" (page 61) from "More On AppleScript Studio" section in previous "AppleScript In Detail" chapter. Includes (not yet fully populated) sections on "Programming Tips" and "Troubleshooting".
	Added material and corrections from previous Release Notes.
	Broke up Watson tutorial chapter "Creating and Connecting the Interface" into two chapters. Note that for a future release, Watson will be renamed Mail Search to match the sample application.
	Broke up Mail Search tutorial chapter "Writing and Debugging the Code" into two chapters.
	Expanded descriptions in "AppleScript Studio Sample Applications" (page 35) to include Archive Maker, Assistant, and Currency Converter applications.
	Small additions and changes to address new or revised features in AppleScript Studio 1.1.
	Many small changes to address developer feedback.
2001-12-01	First release.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.

2006-04-04 | © 2001, 2006 Apple Computer, Inc. All Rights Reserved.