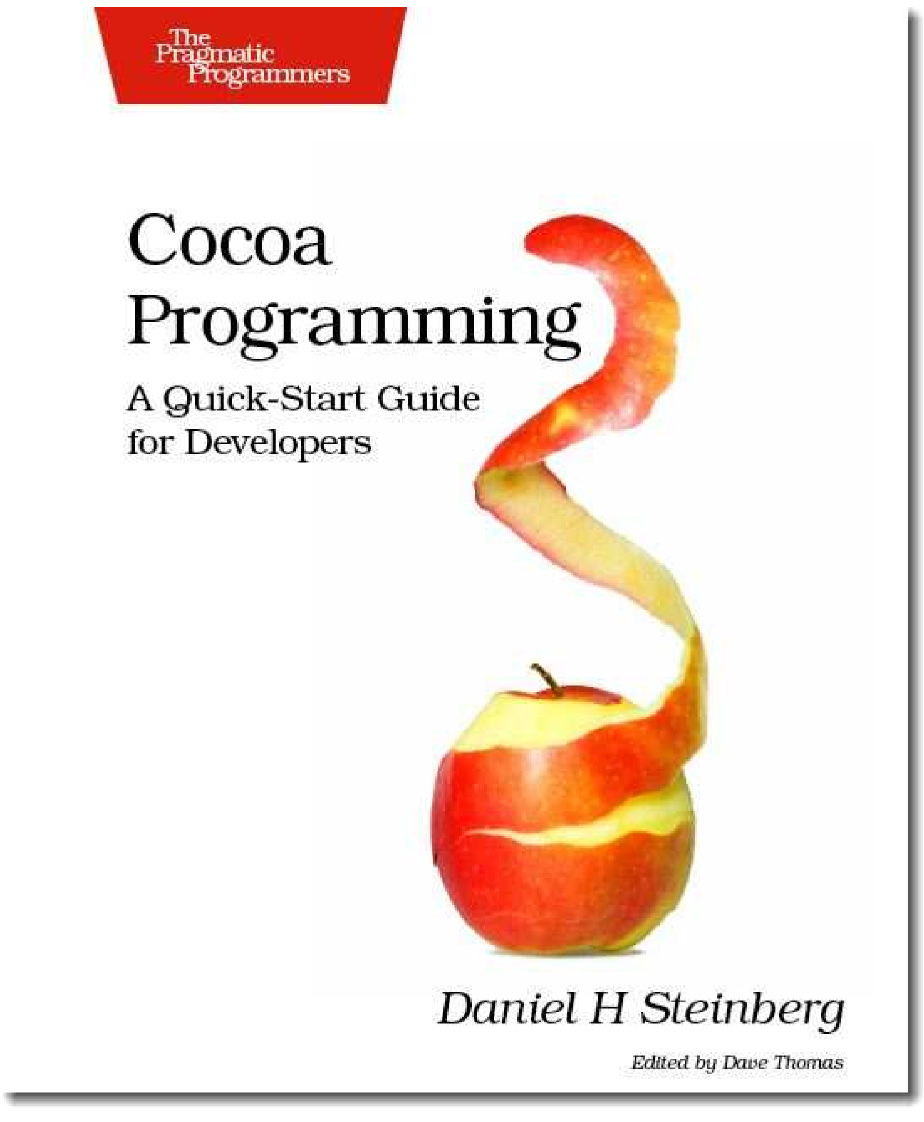
Cocoa Programming

A Quick-Start Guide for Developers



The Pragmatic Programmers

Cocoa Programming

A Quick-Start Guide for Developers

Daniel H Steinberg

Edited by Dave Thomas

ISBN-10: 1-9343563-0-1

ISBN-13: 978-1-9343563-0-2

Printed on acid-free paper.

P1.0 printing, April 2010

Version: 2010-4-5

Chapter 8

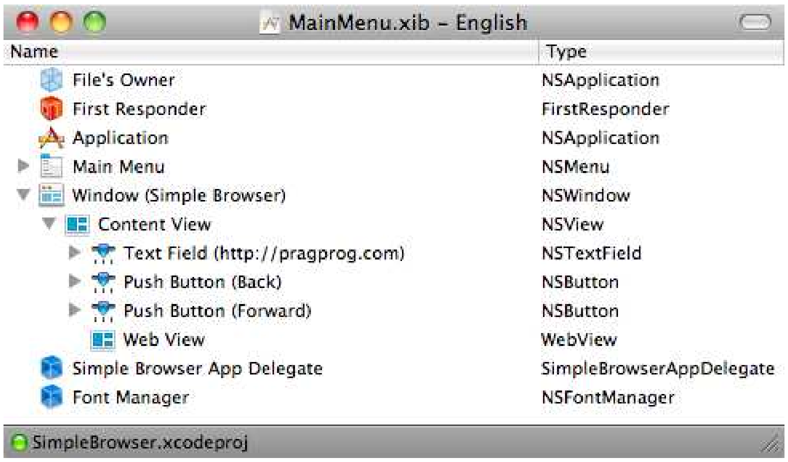
Creating a Controller

You can't accomplish everything you want your application to do just by dragging connections between the visual elements in Interface Builder. On the one hand, it's pretty amazing how easily we created a simple web browser in Chapter 2, *Using What's There*, on page 26 just using visual tools. On the other hand, the browser leaves a lot to be desired. There are some things we're just going to need to code ourselves.

In this chapter, we'll create a controller for our SimpleBrowser example. To keep things simple, we won't have a model—we'll just have a view and a controller. The main point of this example is creating a new class and instantiating it to communicate with objects you created in Interface Builder.

8.1 How We've Created Objects

We have created and connected objects in two different ways. In the SimpleBrowser example, all of our objects are created in the nib. We dragged in buttons, a text field, and a web view, and they joined the SimpleBrowserAppDelegate and all of the other objects that are created in the nib.

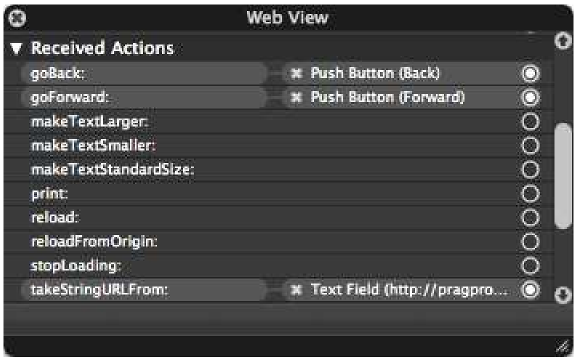


MainMenu.xib – English

Name	Type
File's Owner	NSApplication
First Responder	FirstResponder
Application	NSApplication
▶ Main Menu	NSMenu
▼ Window (Simple Browser)	NSWindow
▼ Content View	NSView
▶ Text Field (http://pragprog.com)	NSTextField
▶ Push Button (Back)	NSButton
▶ Push Button (Forward)	NSButton
Web View	WebView
Simple Browser App Delegate	SimpleBrowserAppDelegate
Font Manager	NSFontManager

SimpleBrowser.xcodeproj

There is effectively no code for this application. All of the connections between the objects have been made in Interface Builder. Here are the actions that the web view receives. It gets the goBack: message from the Back button, the goForward: message from the Forward button, and the takeStringURLFrom: message from the text field.



Web View

▼ Received Actions

goBack:	Push Button (Back)
goForward:	Push Button (Forward)
makeTextLarger:	
makeTextSmaller:	
makeTextStandardSize:	
print:	
reload:	
reloadFromOrigin:	
stopLoading:	
takeStringURLFrom:	Text Field (http://pragpro...

Contrast this with the "Hello, World!" application we built in Chapter 4, *Classes and Objects*, on page 64. There we created a text field in code in our application delegate. We then created a custom Greeter. We instantiated it in code in our app delegate, and all of our communication among the objects we created was in code.

In Chapter 7, *Outlets and Actions*, on page 126, you learned how to connect code that you created with objects that were created in Interface Builder. If you needed to talk to a widget from your code, you created an outlet for that widget in your header file and then connected the outlet to the widget in IB.[1] If you needed a widget to trigger a method that you created in code, you declared this action in the header file and then connected the triggering widget to the action in Interface Builder.

The key to all of this is that there must be an object in your nib of the type that you are adding these outlets and actions to. For now, that will mean that we are creating an instance of our class in Interface Builder.

Let's make this more concrete. We'll create a controller class in code and then create our instance in Interface Builder.

8.2 Creating Our Controller Class

All *classes* are created in Xcode.

Reopen the SimpleBrowser project, or quickly re-create it. In Xcode, choose File > New File... or ⌘N. Choose to create a Cocoa > Objective-C class. I know this doesn't look like a controller class, and there are other options that include the word *Controller*. Don't choose them. What makes this class a controller is how you will configure and use it.

Name your class BrowserController, and make sure that the checkboxes to create BrowserController.h and to target SimpleBrowser are selected. Generally, if you accept the defaults, you should be OK. Click Finish, and save your work.

Our next step is to create an instance of the class and allow it to interact with the GUI elements you've already created. You can instantiate BrowserController using code you write in Xcode or in much the same way we instantiated the GUI elements like NSButton in Interface Builder.

Even though we always write out class code in Xcode, you've seen that we can instantiate them in code or using Interface Builder. We will create objects that belong to the model in Xcode because they don't need to directly know about or communicate with any of the GUI elements. We will create objects that are controller elements in Interface Builder so that we can drag connections between the controllers and the objects they communicate with.

1. I'm using widget to informally refer to one of the GUI elements like buttons, text fields, and so on.

At first, this will feel a bit odd. After all, you are creating an instance of a class that has no visual representation using a tool named *Interface* Builder. In no time, this will feel completely natural to you.

8.3 Creating an Instance of Our Controller in IB

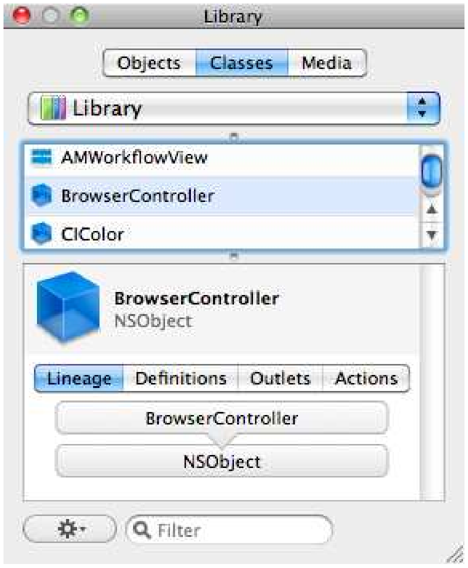
We're now going to create an instance of the BrowserController class in Interface Builder.[2]

When we created instances of our buttons, we just looked in the Library for an NSButton that looked like the one we wanted and dragged it into our window. We can't do that with our BrowserController because there's no way that Interface Builder's Library would contain our BrowserController class—we just made it up. Fortunately, Snow Leopard and Xcode 3.2 have made this a lot easier for us.[3]

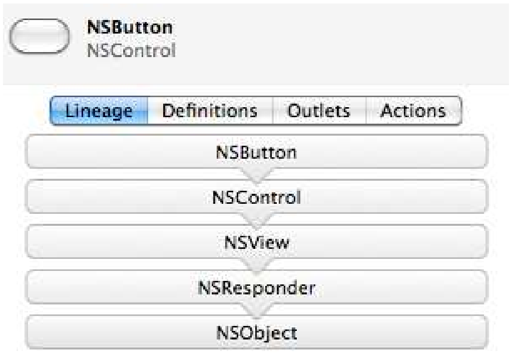
Double-click MainMenu.xib to open the nib file in IB. You aren't interested in the Window view anymore because there is no visual representation of the controller for the end user to see. Instead, bring up the Document window in Interface Builder with the key sequence ⌘0 or Window > Document.

Interface Builder provides us with a representation of our BrowserController object. In IB, go to the Library, and this time choose the Classes tab and look for BrowserController.

2. Actually, the instance isn't really created until the nib is unarchived when the application starts up. We can think of the instance being created at this point in the same way we talked about creating an object in code when we learned to use a call like this [[BrowserController alloc] init].
3. If you are running an earlier version of Xcode, you will need to adjust the directions that follow so that you drag an NSObject into the Document window and change its type using the Identity inspector.



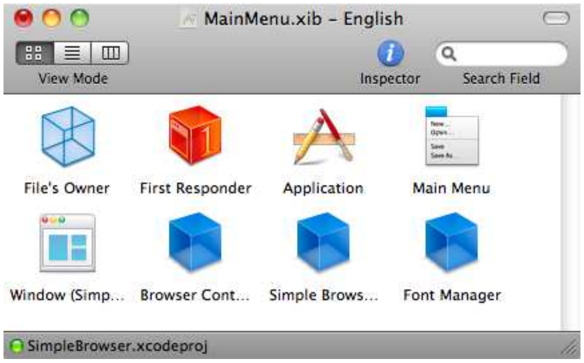
Our BrowserController class has a simple lineage since it directly extends NSObject. If you look at the lineage for a class like the NSButton, you'll see a much deeper hierarchy.



In addition to the Lineage tab, there are also tabs for looking at the class's outlets and actions and also where the class is defined. You'll notice the + and - in the bottom-left corner of both the Outlet and Action tabs. Do not add or remove outlets or actions in Interface Builder. I know it seems as if it would be easier, but changes you make in IB are not picked up by Xcode. Make your changes in the code, and let those changes be picked up in IB.

Back in our example, drag the BrowserController into the Document window, and drop it. Congratulations! You've just created an instance of

BrowserController in the nib. Here is the icon view of your Document window with the newly added instance of the BrowserController:

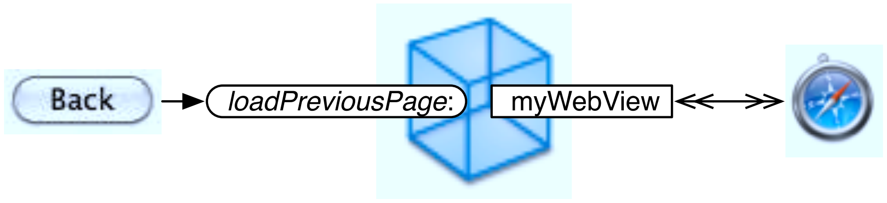


I prefer using the list view but wanted to show you this view in case you like it better.

8.4 Declaring an Outlet and an Action

Right now the Back button is connected to the web view's goBack: method. Let's insert the BrowserController in between. We're going to need a method in the controller that we use for loading the previous web page. It will be called by the Back button and will, in turn, have to send the message goBack: to the web view. This means we'll need an outlet for the web view and an action for the method.

Back in Xcode, add an IBAction named loadPreviousPage: to BrowserController.h. You'll also need to add an IBOutlet named myWebView using an instance variable and a property.[4] The myWebView variable is a pointer to a WebView object.



4. If you are using and are targeting 64-bit, you can create your property without declaring the instance variable.