

Extracted from:

Cocoa Programming

A Quick Start Guide for Developers

This PDF file contains pages extracted from Cocoa Programming, published by the Pragmatic Bookshelf. For more information or to purchase a paperback or PDF copy, please visit <http://www.pragprog.com>.

Note: This extract contains some colored text (particularly in code listing). This is available only in online versions of the books. The printed versions are black and white. Pagination might vary between the online and printer versions; the content is otherwise identical.

Copyright © 2009 The Pragmatic Programmers, LLC.

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, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

The
Pragmatic
Programmers

Cocoa Programming

A Quick-Start Guide
for Developers



Daniel H Steinberg

Edited by Dave Thomas



Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and The Pragmatic Programmers, LLC was aware of a trademark claim, the designations have been printed in initial capital letters or in all capitals. The Pragmatic Starter Kit, The Pragmatic Programmer, Pragmatic Programming, Pragmatic Bookshelf and the linking *g* device are trademarks of The Pragmatic Programmers, LLC.

Every precaution was taken in the preparation of this book. However, the publisher assumes no responsibility for errors or omissions, or for damages that may result from the use of information (including program listings) contained herein.

Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at

<http://www.pragprog.com>

Copyright © 2009 Daniel H. Steinberg.

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, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

Printed in the United States of America.

ISBN-10: 1-9343563-0-1

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

Printed on acid-free paper.

B1.07 printing, May 29, 2009

Version: 2009-5-29

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 can create a simple web browser just using visual tools. On the other hand, the browser leaves a lot to be desired. There are some things you're just going to need to code up yourself.

Cocoa programming separates the application logic from the look and feel using Model-View-Controller (MVC). For the model, we'll create the application logic in Objective-C using Xcode. You've already seen that we create the view using Interface Builder.

The controller is the bridge between the model and the view. When the user clicks on a button or types in a text field or does anything to the view, the controller communicates these actions to the model. Similarly, when the model changes, the controller updates the view so that the changes are visible to the user.

The controller has to have a foot in each world. There is a class file that you use to create methods and send messages to the model or the view. There is also a visual representation of the controller that lives in Interface Builder that you use to wire the controller's code to the visual components you create in IB. It's sort of like having the real version of the controller living in code and its Second Life® avatar living in IB.

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. In a way, most of the application logic is being managed by the web view. Just as with the SimpleBrowser example in Chapter 2, *Using What's There*, on page 24, we'll create this controller

in small discrete steps to explain all of the new concepts. You'll combine and rearrange once you understand what you're doing.

4.1 Reusing your NIB

You can continue to work with the SimpleBrowser project or you can create a new project and copy over your NIB file. I created a new Cocoa Application and named it “SimpleBrowserWithController”. I then used the Finder to locate the NIB file from SimpleBrowser. It's at `.../SimpleBrowser/English.lproj/MainMenu.xib`. Copy `MainMenu.xib` and paste it over `.../SimpleBrowserWithController/English.lproj/MainMenu.xib`.

Back in Xcode you can double-click on `MainMenu.xib` and Interface Builder will open up with the configuration you created in the previous project. Before you forget, you are going to have to add the Web Kit framework again or you will get errors building in Xcode.

4.2 Creating your controller class

All classes are created in Xcode.¹ From Xcode choose `File > New File ...` or ⌘N. Choose to create a Cocoa > Objective-C class. The description should say that you are creating “An Objective-C class file, with an optional header which includes the `<Cocoa/Cocoa.h>` header.” 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 `SimpleBrowserController` and make sure that the checkboxes to create `SimpleBrowserController.h` and to target `SimpleBrowserWithController` are checked.² Generally, if you accept the defaults you should be ok. Click `Finish`.

You've now generated a header file `SimpleBrowserController.h` and also an implementation file `SimpleBrowserController.m`. Here's the header file with two comments I've inserted to help our discussion:

1. If you have used Xcode and IB in the past you'll note that this is a change. You'll also see, in this chapter, that you no longer have to drag your header files back into IB every time you make a change. The synchronization is done for you.

2. I'm going to assume you are working with a new Cocoa Application project that you've named “SimpleBrowserWithController”.

Organizing Files

At the left side of your Editor window you should see a section labeled “Groups & Files”. You may need to use the disclosure triangle next to your application’s name to reveal folders labeled “Classes”, “Other Sources”, “Resources”, “Frameworks”, and “Products”.

When you create a new class in Xcode, the implementation and header files will appear in whatever directory is selected. If none is selected they will appear in the top level under the application name. There is no correspondence between where the files actually live on the disk and where they appear in this organizer. The organizer works more like playlists in iTunes than like mailboxes in Mail.

Create other folders if you like, but for small projects you want to keep your source code in the “Classes” folder. Either select “Classes” before you create a new class or drag the files into the folder afterwards.

Download [CreatingAController/SimpleBrowserController1/SimpleBrowserController.h](#)

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

@interface SimpleBrowserController : NSObject {
    // you'll declare instance variables here
}
    // you'll declare methods here
@end
```

The header file contains the public interface for the SimpleBrowserController class. You use it to tell other people how they can interact with your class. At the top of the file you often import the header files of other classes your class might want to use. In this case Xcode has already included the directive to import Cocoa.h which includes the header files for all of the Cocoa classes you might need to use.

Everything between @interface and @end is the description of the public interface for the SimpleBrowserController class.

SimpleBrowserController : NSObject indicates that the SimpleBrowserController class directly extends the root class NSObject. WebView, in contrast, is a subclass of NSView which in turn is a subclass of NSResponder which is a subclass of NSObject (see the (as yet) unwritten *fig.readingobjc.docformat*).

Unless you specifically override it, you benefit from inheriting the behavior of any of your super classes. The behavior that is common to all objects is specified in the root class `NSObject`. You will sometimes have to look in the documentation of a superclass to find methods that are available to objects created from your class.

In the header file, the comments I've added to the template code show that you add the declaration for your instance variables inside the curly braces and you declare your methods between the closing brace and the `@end`.

Here's `SimpleBrowserController.m`, the implementation³ file you just generated,

[Download](#) `CreatingAController/SimpleBrowserController1/SimpleBrowserController.m`

```
#import "SimpleBrowserController.h"

@implementation SimpleBrowserController

@end
```

The file begins by importing the header file for `SimpleBrowserController`. Other than that you will see beginning and end markers for the class implementation. Notice that in this file you don't specify that the `SimpleBrowserController` inherits from `NSObject`.

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 `SimpleBrowserController` using code you write in Xcode or in much the same way we instantiated the GUI elements like `NSButton` in Interface Builder. We will create objects that belong to the model in Xcode as 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.

3. Note the suffixes of the two files `SimpleBrowserController.h` and `SimpleBrowserController.m`. The "h" is for header and "m" is for implementation. You'll find this and other fun facts in the Objective-C FAQ at <http://www.faqs.org/faqs/computer-lang/Objective-C/faq/>.

4.3 Creating an instance of our controller in IB

We're now going to create an instance of `SimpleBrowserController` in Interface Builder.⁴

When we created instances of our buttons we just looked in the Library for the `NSButton` that looked like the one we wanted and dragged it into our window. We can't do that with our `SimpleBrowserController` for a couple of reasons.

- There is no way that Interface Builder would know about our `SimpleBrowserController` class—we just made it up. So we'll have to grab a generic `NSObject` from the Library. You will find controllers to choose from as well but remember that `SimpleBrowserController` is a subclass of `NSObject` so that's what you'll use initially to represent it.
- We'll need a way of letting Interface Builder know that this object is not just an `NSObject` but that it is actually an instance of `SimpleBrowserController`.
- The `SimpleBrowserController` object won't be visible to the users. We don't want to drag our object representation into the window the way we dragged and positioned buttons, the text field, and the web view. We need another place to put it.

Let's start with that last point. Double-click on `MainMenu.xib` (under Resources in the side menu) to open the NIB file in IB. You aren't interested in the Window view any more as 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`. You can see the higher level visible and non-visible components of `SimpleBrowserWithController`.

4. 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 will talk about creating an object in code when we learn to use a call like this `[[SimpleBrowserController alloc] init]`.



We need something to represent our SimpleBrowserController object in the Document window. Go to the Library and look in the Cocoa > Objects and Controllers group. Click on NSObject (which is called Object in the list) and drag it into the Document. As before, you will be tempted to drag one of the controllers in but don't—choose Object.

Now we need to change the object from being an instance of NSObject to being an instance of SimpleBrowserController. Click on the Object icon you just dragged into the Document and look at the Identity Inspector for it by clicking the *i* icon in the blue circle at the top of the inspector window.



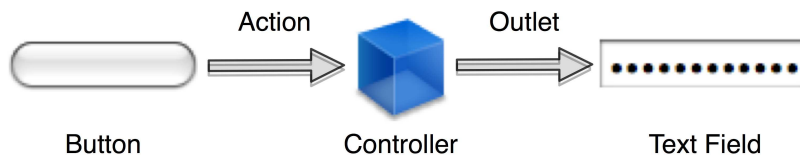
On your screen you'll see the Class Identity listed as NSObject. In its place enter SimpleBrowserController as shown in the figure. Enter, or scroll down and select from the list. Save your work. You have created a controller object.

4.4 Actions and Outlets

There are basically two ways in which the controller connects to UI elements:

- **Actions**—controller methods used when an element such as a button wants to initiate an action performed by the controller.
- **Outlets**—controller instance variables that point to the UI elements the controller needs to send messages to.

Actions and Outlets are specifically designed for connections created in Interface Builder. Here's a look at the basic flow.



When a user presses a button a message is sent to a specified target to initiate a specific action. You create this target action in a controller and you make the connection in interface builder. The action is just a method that will get called when the button is pressed.

There are times the controller is going to need to communicate with an object you created using IB. One way is to give the controller a handle to the object. In a minute we'll give our controller an outlet which is a text field. In other words, the controller has an instance variable that points to the text field. Just like a wall socket, the outlet is a place in the controller where the visual element plugs in to.

Our SimpleBrowserController doesn't have any actions or outlets yet. In the next section you will modify your header file to add some. Once you save your changes in Xcode, Interface Builder will automatically pick up these changes and allow you to drag and drop connections to and from the other elements you have created.

The signature of the Actions look a lot like the `goBack:` method we examined in Section 3.4, *Methods with Arguments*, on page 48:

The Pragmatic Bookshelf

The Pragmatic Bookshelf features books written by developers for developers. The titles continue the well-known Pragmatic Programmer style and continue to garner awards and rave reviews. As development gets more and more difficult, the Pragmatic Programmers will be there with more titles and products to help you stay on top of your game.

Visit Us Online

Cocoa Programming's Home Page

<http://pragprog.com/titles/dscpq>

Source code from this book, errata, and other resources. Come give us feedback, too!

Register for Updates

<http://pragprog.com/updates>

Be notified when updates and new books become available.

Join the Community

<http://pragprog.com/community>

Read our weblogs, join our online discussions, participate in our mailing list, interact with our wiki, and benefit from the experience of other Pragmatic Programmers.

New and Noteworthy

<http://pragprog.com/news>

Check out the latest pragmatic developments in the news.

Buy the Book

If you liked this PDF, perhaps you'd like to have a paper copy of the book. It's available for purchase at our store: pragprog.com/titles/dscpq.

Contact Us

Phone Orders:	1-800-699-PROG (+1 919 847 3884)
Online Orders:	www.pragprog.com/catalog
Customer Service:	orders@pragprog.com
Non-English Versions:	translations@pragprog.com
Pragmatic Teaching:	academic@pragprog.com
Author Proposals:	proposals@pragprog.com