

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 © 2010 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 © 2010 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.

P1.0 printing, April 2010

Version: 2010-4-5

Customizing with Delegates

Stuff happens.

In fact, lots of stuff is happening as your end user enters URLs and clicks buttons and links in your SimpleBrowser. The most basic type of event is target-action. The user clicks a button, and an action is sent to a target. You've learned two ways of working with these events: you can use Interface Builder to directly wire the object sending the message to the target object that will perform the action, or you can create a controller.

But once you enter a URL or use the Back or Forward button, additional events and messages are flying by that you don't even see. For example, when you load a page in Safari, you've probably noticed that the title of the new page is displayed above your toolbar before the page actually loads. Meanwhile, you can see the progress of the page being loaded by the blue bar that works its way across the text field containing your URL. There are messages being sent that Safari is able to capture and respond to. What about us?

There are plenty of messages zipping by us all the time. In this chapter, you'll learn to listen for them and how to respond to them. We'll display the title of the web page we're loading and fix the buttons and the URL while we're at it. Delegates will let us customize behavior for a class without creating a subclass.

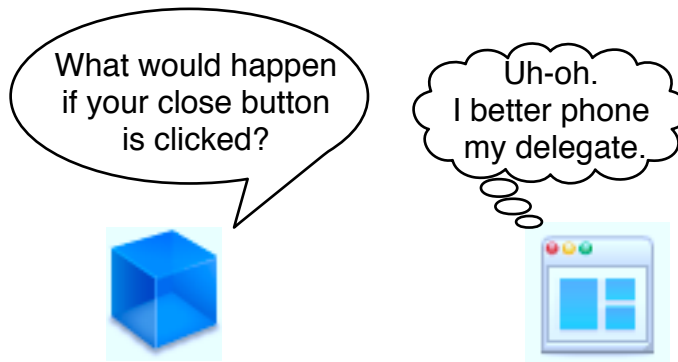
9.1 Understanding Delegates

Before we apply delegates to our browser example, we will use the `NSWindow` class to explore how delegates work.

Imagine that our `NSWindow` object is appearing on a Cocoa game show. If you look at the documentation for `NSWindow`, you will see more than 200 methods listed as tasks that our window might be asked to perform.

Our window object is pretty confident that it knows how to respond to most of the messages it might be sent. For example, it knows how to respond to the message `setShowsToolbarButton:`. If our window has a toolbar, then passing in `YES` displays the toolbar button, and passing in `NO` hides the toolbar button. If our window doesn't have a toolbar, then calling this method doesn't do anything.

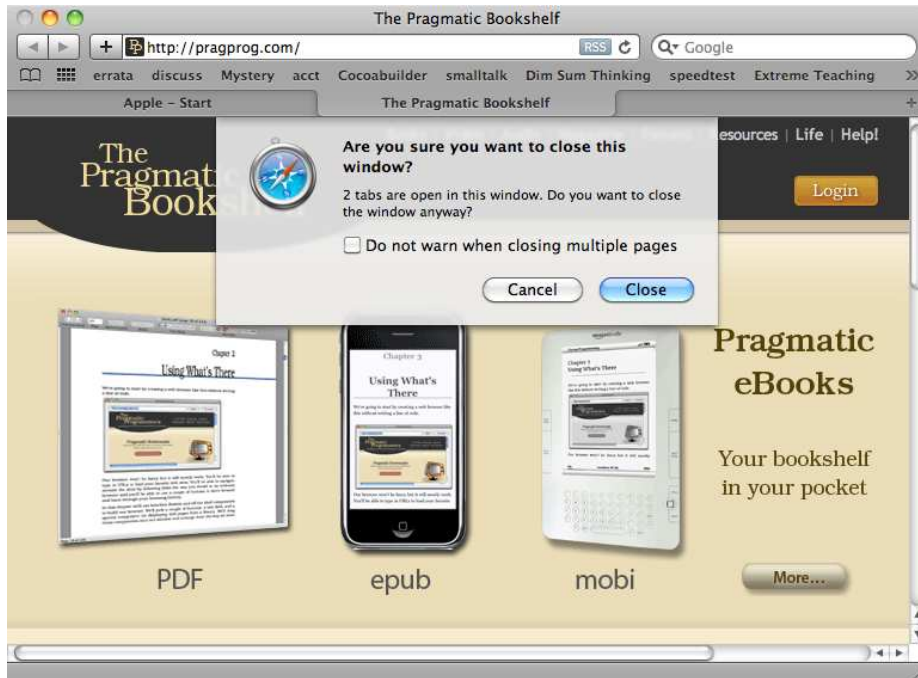
On the other hand, there are some tasks that our window doesn't know how to react to. For example, what if our window is asked this on the talk show?



The answer is not as simple as you may think. Usually, when someone clicks the red button, the window will close, and the application will continue to run. Even though this is the default behavior, it is not the only possibility. For example, if you open System Preferences and click the red button, the window closes, and this time the application quits too.

You'll see a third type of behavior if you open a web page in Safari and then create a new tab (with tabbed browsing enabled, select `File > New Tab` or `⌘T`). Open a web page in the new tab, and then hit the red button.

Even though you've clicked the same red button in the same application, the window doesn't automatically close when you have multiple tabs open. Instead, it asks you, "Are you sure you want to close this window?"



These variations present a dilemma for our window. What should it answer if it is asked by the game show host, “What would happen if I click your red button”?

The window wants to answer “It depends,” but that’s not an acceptable answer in a game show or in a running Cocoa application. So, the window chooses to get help answering this question by phoning a friend.

Here’s the way the rules work for this game. The window identifies all the messages that it might need help with ahead of time and labels those as delegate methods.¹ In the case of the `NSWindow` class, here are all of the delegate methods:

```

window:shouldDragDocumentWithEvent:from:withPasteboard:
window:shouldPopUpDocumentPathMenu:
window:willPositionSheet:usingRect:
windowDidBecomeKey:
windowDidBecomeMain:
windowDidChangeScreen:
windowDidChangeScreenProfile:
windowDidDeminiaturize:
  
```

1. In this chapter, we’re using protocols and delegates that Apple created. We’ll create our own delegate and protocol in Chapter 12, *Creating Protocols for Delegation*, on page 205.

```

windowDidEndSheet:
windowDidExpose:
windowDidMiniaturize:
windowDidMove:
windowDidResignKey:
windowDidResignMain:
windowDidResize:
windowDidUpdate:
windowShouldClose:
windowShouldZoom:toFrame:
windowWillBeginSheet:
windowWillClose:
windowWillMiniaturize:
windowWillMove:
windowWillResize:toSize:
windowWillReturnFieldEditor:toObject:
windowWillReturnUndoManager:
windowWillUseStandardFrame:defaultFrame:

```

The delegate methods come in three flavors: something will happen, something did happen, and something should happen. You implement the will version to change the behavior before an action happens and the did version to respond after the action happens. The should version returns a `BOOL` that allows you to cancel an action if you determine it shouldn't happen.

Just like the game shows where a contestant can “call a friend,” each window object gets to designate a single friend that can be called when any of these methods is called. That friend is the delegate.

We don't really expect one friend to be able to respond to all of these messages, but our window must choose only one delegate ahead of time. Unfortunately, the window can't say, “I know who would know how to answer that question” and decide who will be called at runtime. The delegate is chosen before the question is asked.

Before the game begins, we preinterview the delegate. This way, we know which messages the delegate can handle and which ones it can't. In Cocoa terms, the delegate may implement none, any, or all of the delegate methods for an object. If a delegate has not implemented a method, it will not be called to handle that message. The window will have to come up with its own answer.

These rules might seem complicated—but a quick example should clarify the situation.

9.2 The Default Window Behavior

Create a new project in Xcode that is a Mac OS X > Application > Cocoa Application called WindowDressing. We're going to use a delegate to change the default behavior of the green and red buttons.

Click Build & Run. When you click the green button, the window should zoom up to fill most of the screen. Click it again, and the window will zoom to its original size. Click the red button and the window closes, but the application doesn't quit. Once you click the red button, there's not much more you can do. We can't create a new window so we have to quit this application anyway.

So far, we've seen the default behavior when the green or red button is clicked. This is the behavior the window provides when it isn't allowed to phone a friend. In the next section, we'll provide alternate behavior using a delegate.

9.3 Turning the Background Red

Let's change the behavior so that when the user clicks the red button, the background of the window turns red. We'll follow these steps:

1. We'll need to create a new class that we can instantiate to create the window's delegate.
2. We'll designate that object as the window's delegate.
3. We'll identify the delegate method we need to implement.
4. Finally, we'll implement the method.

The second and third steps are where people often make small, hard-to-trace mistakes. It's easy to forget to connect the window to its delegate in Interface Builder. Also, make sure that you save your nib file after you make a change. When you build your project in Xcode, you will be asked whether you want to save any source files that have been changed, but you may have unsaved changes in Interface Builder as well.

I guarantee this is going to bite you at some point. If you have been bouncing back and forth between Xcode and Interface Builder and your project isn't behaving the way you expect, take a look at your nib's Document window. If you see a dark dot in the middle of the red close button, then your file is dirty and needs to be saved.



If there are no unsaved changes, then the red close button should be clear like the yellow minimize button and green zoom button. It's a subtle hint used across Mac OS X apps and one worth watching for.



You also need to be careful to spell the delegate method name exactly as it appears in the `NSWindow` documentation. When we worked with buttons, we were free to assign any object as the target and make up the name of the method. Now with delegates, the name of the method is chosen for you.

In Xcode, create a new class file by selecting `File > New File...` or `⌘N`. Choose the template `Mac OS X > Cocoa Class > Objective-C class`, and name it `WindowHelper`.

Just like the previous chapter, we'll now create a representation of this class in Interface Builder. Double-click `MainMenu.xib` to open it in Interface Builder. Find `WindowHelper` under the `Classes` tab in the Library, and drag it into the Document window.² Control-click the `Window` object—not the `WindowHelper` object—inside the Document window to bring up this connections window.



Drag from the circle next to the `delegate` outlet to the `WindowHelper` object inside the Document window. This sets the window's delegate to be an instance of the `WindowHelper`. Save your work, and quit Interface Builder.

Back in Xcode, you don't really have to make any changes to the `WindowHelper` header file. You've made the connection to the delegate in Interface Builder. As a developer, you know the signature of the methods you can implement. At runtime the system knows the messages it

2. You'll find it a lot quicker if you start typing `Win` into the search field at the bottom of the Library window.

can send. You can, however, make the intent of the WindowHelper clearer by indicating that it implements the `NSWindowDelegate` protocol.

A protocol declares a collection of methods that a class may or may not implement. You declare one or more protocols at the top of your header file between angle brackets like this:

[Download](#) Delegates/WindowDressing2/WindowHelper.h

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

```
► @interface WindowHelper : NSObject <NSWindowDelegate> {
}
@end
```

If you check the docs for the `NSWindowDelegate` protocol, you'll see that all of the methods are listed as *optional*. In other words, you are free to implement only the ones you need.

There are only two delegate methods that have anything to do with a window closing. There's `windowShouldClose:` and `windowWillClose:`. The `windowShouldClose:` method is called when the user clicks the red button and gives you an opportunity to say "Don't close the window." The `windowWillClose:` method is called just before the window closes. That's too late for our needs.

We'll use the `windowShouldClose:` method. Because it's so easy to introduce typos that are hard to detect and debug, I often go to the docs and cut and paste the method signature.

The `windowShouldClose:` method needs to do two things. It has to set the window's background color to red. This time, the sender is the window, so we can just tell the sender to set its background color to red.

The method also has to return `NO` so that the window doesn't close. If you return `YES`, the window will close. Here's `WindowHelper.m`:

[Download](#) Delegates/WindowDressing2/WindowHelper.m

```
#import "WindowHelper.h"
```

```
@implementation WindowHelper
- (BOOL)windowShouldClose:(id)sender {
    [sender setBackgroundColor:[NSColor redColor]];
    return NO;
}
```

```
@end
```

Click Build & Run, and the window will now turn red when you click the red button. More specifically, when you click the red button, the message `windowShouldClose: message` is sent to the object you've designated as the delegate for the window object. Quit `WindowDressing` with `⌘Q` or by choosing Stop from Xcode.

9.4 Exercise: Turning the Background Green

Find the method that is called when someone clicks the green button to zoom your window to another frame size. Implement this method so that the window does not change size and so that the background of the window turns green when the green button is clicked.

9.5 Solution: Turning the Background Green

All that you need to do is implement the `windowShouldZoom:toFrame:` method. The body should look almost exactly like `windowShouldClose:`.

[Download](#) Delegates/WindowDressing3/WindowHelper.m

```
- (BOOL)windowShouldZoom:(NSWindow *)window toFrame:(NSRect)newFrame {
    [window setBackgroundColor:[NSColor greenColor]];
    return NO;
}
```

Click Build & Run. You can now change the window color to red and green by clicking the corresponding buttons.

9.6 Application Delegate

We've actually been using delegates since our very first project. The Cocoa Application template introduced in Xcode 3.2 includes an application delegate. Now that you understand the idea in the context of windows, you should be able to see how it applies to applications.

When an application launches, there are lots of things that must happen, and Apple takes care of that for you. But there are also things that are special to your application that Apple can't possibly know about. If there are particular things that you want to occur after the application finished launching, you implement the application delegate's `applicationDidFinishLaunching:` method.

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