

Extracted from:

# Programming Cocoa with Ruby

---

## Create Compelling Mac Apps Using RubyCocoa

This PDF file contains pages extracted from Programming Cocoa with Ruby, 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

# Programming Cocoa with Ruby

Create Compelling Mac Apps  
Using RubyCocoa



*Brian Marick*

Edited by Daniel H Steinberg

The Facets  of Ruby Series



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 Brian Marick.

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: 1-934356-19-0

ISBN-13: 978-1-934356-19-7

Printed on acid-free paper.

P1.0 printing, July 2009

Version: 2009-8-7

# One Good App Observes Another

---

Now that Fenestra has a (barely) tolerable interface, it's time to work on the code behind it. We'll use Cocoa's notification system for communication between Fenestra and some other app. Because notifications are widely used in Cocoa apps, I'll describe them in more detail than is needed for just this one app.

### 4.1 Notifications Within an App

The version of our app in Section 3.6, *Overriding Window Behavior with a Delegate*, on page 62, works because some object<sup>1</sup> follows the main window's delegate link to a Controller, notices that it defines `windowWillClose`, and calls that method, giving it a chance to make the app exit.

Controller can learn about the window closing in another way. It can subscribe to notifications from the `NSWindow`:

[Download](#) `fenestra/autoclose-with-notifications/Controller.rb`

```
def awakeFromNib
  center = NotificationCenter.defaultCenter
  center.addObserver_selector_name_object(self, :windowWillClose,
                                          'NSWindowWillCloseNotification',
                                          @logWindow)

  record('')
end
```

In words, our Controller is saying, “Hey! Default notification center! At some point, the object I know as `@logWindow` might announce that it's going to close. If so, send the `windowWillClose` message to a particular

---

1. It's not actually the main window's `NSWindow` itself, but it might as well be.

object (namely, me).” The `windowWillClose` method is unchanged from the one in the previous version (found on page 63):

[Download](#) fenestra/autoclose-with-notifications/Controller.rb

```
def windowWillClose(notification)
  NSApp.terminate(self)
end
```

I could have named the method something else (which you can't with a delegate), but when it comes to windows, closing delegation and observing are just different ways of getting the same information to our Controller, so using the same name seems appropriate. (Even `windowWillClose`'s argument is the same. It's an `NSNotification` object, described in Section 4.1, *The Finer Points of Notifications*, on the following page.)

You can see the new version working by first using IB to turn off the window's delegate outlet (click the little *x* if the outlet shows a connection) and then building and running.

## Delegation vs. Notification

You have now seen two different code designs for window closing. You might ask, which is better? Since they both do the same thing, I embrace my inner slacker and ask two questions:

- Which is less work today?
- Which will be less work in the future?

I personally place more weight on the first question because I get its answer right more often.

Setting a delegate requires drawing a line in Interface Builder. Adding a notification means typing code in `awakeFromNib`. For me, delegation wins.

Looking to the future, I can imagine myself adding another window to the app. After that, Fenestra should behave like other multiwindow Mac apps: closing a window just closes the window. If you want to exit, do that explicitly.

I know my imaginary future self all too well: he doubtless will have forgotten both how and where I implemented the current behavior. If I used delegation, all my future self will have to do is look for the window's delegate in IB, hop to that class, and find method `windowWillClose`. If I used notifications, finding the code would be more work. First, I'd have the wasted work of checking for the delegate; on top of that, I'd have to grovel around in the code to find a method that wouldn't necessarily even be called `windowWillClose`.

## Try This Yourself

Have `windowWillClose` print *"I'm here!"* to the console. Either puts or NSLog work fine. Then:

1. Build and run the app. Close the window (⌘-W).
2. Run the app again. Quit the program (⌘-Q).
3. Reestablish the delegate link between the main window and the Controller, but continue to make the Controller an observer of the `@logWindow`. Build and run. Exit by closing a window.

What do you think is happening?

I think that `NSApplication`'s `terminate` method closes all open windows. In our case, `terminate` is called because an open window is closing. So, `terminate` blithely closes that window again. `windowWillClose` again calls `terminate`. Fortunately, `terminate` is smart enough not to go any further down the rat hole.

It also seems that delegation to `windowWillClose` is independent of notification delivery. So, in the third case, `terminate` is called three times: once because of delegation, once because of notification, and once because `terminate` closes all windows.

## The Finer Points of Notifications

“Don't care” values

In the previous example, the code asked to hear about notifications named `NSWindowWillCloseNotification`. A `nil` argument asks to hear about notifications with any name. Try that, printing the notifications with `puts notification`; then try various window operations (such as minimizing and hiding) to see what notifications get sent. To see a complete list, use the `NSWindow` class reference.

If you use `nil` for the object to observe, you'll observe all objects that send a particular named notification. In our case, that's not interesting, since only windows send `NSWindowWillCloseNotifications`, and we have only one window. You can, however, give `nil` as both the name and object arguments. Then you see *all* notifications from *all* objects. Try that to see how many notifications are sent in even the simplest Cocoa applications. (If you use `windowWillClose` to print out the notifications, comment out the line that calls `terminate`.)

`userInfo` arguments

If you tried the change in the previous paragraph, you probably saw output with extra information. Here is the notification, for

example, that comes from hitting `Tab` or `Return` to finish editing in a text field:

```
NSConcreteNotification 0x3c9330 {name = NSControlTextDidEndEditin↵
gNotification; object = <NSTextField: 0x3e15c0>; userInfo = {
    NSFieldEditor = <NSTextView: 0x2307490>
    Frame = {{2.00, 3.00}, {271.00, 17.00}}, Bounds = {{0.00, 0.0↵
0}, {271.00, 17.00}}
    Horizontally resizable: YES, Vertically resizable: YES
    MinSize = {271.00, 17.00}, MaxSize = {40000.00, 40000.00}
;
    NSTextMovement = 16;
}}
```

Each notification can pass along an `NSDictionary`. `NSDictionary` is Cocoa's equivalent of Ruby's `Hash`: a collection of key/value pairs. When printed, an `NSDictionary` looks something like a hash, but not exactly. Keys and values are separated by `=,` not `=>`, and strings aren't enclosed in quotes.

If you were writing code in Objective-C, you'd retrieve `NSDictionary` values like this:

```
[dictionary objectForKey: key]
```

You can do the same in Ruby if you want:

```
dictionary.objectForKey(key)
```

But ordinary hash notation also works:

```
dictionary[key]
```

Beware, though: not all `Hash` methods will work on an `NSDictionary`.

### The name and sender

A notification contains its name and a pointer to the object that sent it. They're retrieved like this:

```
notification.name
notification.object
```

### Sending notifications

You send a notification like this:

[Download](#) notifications/examples/within-process-userinfo.rb

```
Center.postNotificationName_object_userInfo("notification name",
self,
{'string' => 'world',
'int' => 5,
'array' => ARGV})
```

(To save horizontal space, I've defined constant `Center` to be the same `NSNotificationCenter defaultCenter` we've already seen.)

If you have no `userInfo` to add, use a slightly different method:

```
Download notifications/examples/within-process.rb
```

```
Center.postNotificationName_object("notification name", self)
```

In cases where receivers aren't expected to care which object sent the notification, programmers sometimes use the *object* argument to send data that more properly should go into *userInfo*. That is, rather than writing this:

```
Center.postNotificationName_object_userInfo("got argv",
                                           self,
                                           { "argv" => ARGV })
```

... they'll write the following:

```
Center.postNotificationName_object("got argv", ARGV)
```

## Conversions

When you create a notification, you'll likely use Ruby objects such as strings, integers, arrays, and nested hashes to fill its *userInfo*. When it's received, though, the Ruby objects have all been converted to their Objective-C equivalents: `NSStrings`, `NSNumber`s, `NSArray`s, and nested `NSDictionary` objects:

```
% ruby within-process-userinfo.rb with args
=== Looks innocent enough when you 'to_s' it:
NSConcreteNotification 0x57ae90 {name = notification name; object =
  t = <Sender: 0x2a9ce0>; userInfo = {
    array =      (
      with,
      args
    );
    int = 5;
    string = world;
  }}
}}

```

```
=== ... but those are not simple Ruby objects:
#<NSDictionary {#<NSString "int">=>#<NSNumber 5>, #<NSString "array">=>#<NSArray [#<NSString "with">, #<NSString "args">], #<NSString "string">=>#<NSString "world">}>
```

For even more about notifications, see Apple's *Introduction to Notification Programming Topics* [[App08q](#)].



# 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

---

### Programming Cocoa with Ruby's Home Page

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

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, new titles and other offerings.

## Buy the Book

---

If you liked this eBook, perhaps you'd like to have a paper copy of the book. It's available for purchase at our store: [pragprog.com/titles/bmrc](http://pragprog.com/titles/bmrc).

## Contact Us

---

Online Orders:	<a href="http://www.pragprog.com/catalog">www.pragprog.com/catalog</a>
Customer Service:	<a href="mailto:support@pragprog.com">support@pragprog.com</a>
Non-English Versions:	<a href="mailto:translations@pragprog.com">translations@pragprog.com</a>
Pragmatic Teaching:	<a href="mailto:academic@pragprog.com">academic@pragprog.com</a>
Author Proposals:	<a href="mailto:proposals@pragprog.com">proposals@pragprog.com</a>
Contact us:	1-800-699-PROG (+1 919 847 3884)