Extracted from:

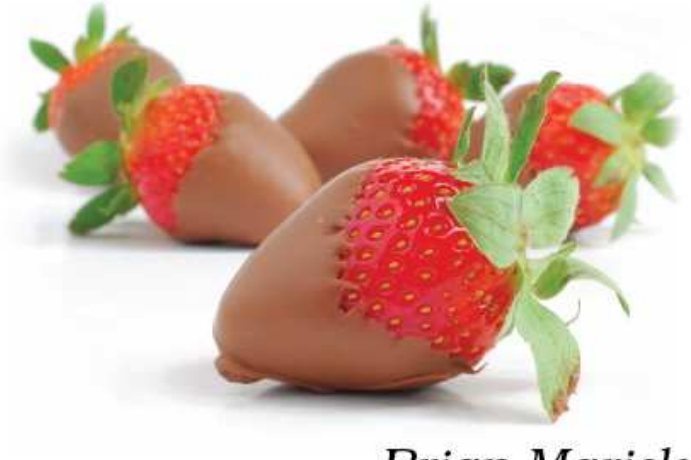
Programming Cocoa with Ruby

Create Compelling Mac Apps Using RubyCocoa

The Pragmatic Programmers

Programming Cocoa with Ruby

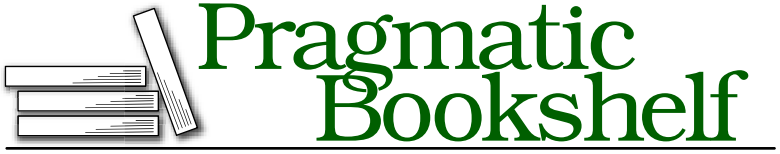
Create Compelling Mac Apps Using RubyCocoa



Brian Marick

Edited by Daniel H Steinberg

The Facets of Ruby Series



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

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

6.3 Reacting to Button State

When a button is clicked, it invokes an action method. That action method can query the button for its state and act appropriately. Here's such code:

Download fenestra/reshaped-but-gutted/AppChoiceController.rb

```
ib_action :chooseOrHeal

def chooseOrHeal(sender)
  NSLog("AppChoiceController button pushed.")
  if @button.state == NSOnState
    NSLog("Fenestrate '#{@comboBox.stringValue}'.")
  else
    NSLog("Heal.")
  end
end
```

Before you can see that working, you'll need to connect the button to the chooseOrHeal method, using Interface Builder. (Either you can drag to the button from the AppChoiceController's chooseOrHeal received action or you can drag from the button's selector sent action to the AppChoiceController.)

For the complete description of buttons, see the NSButton class reference and *Button Programming Topics for Cocoa* [App08e].

6.4 Using Nibs to Avoid Dependencies

The AppChoiceController connected to a TranslatorEnlister is shown in Figure 6.1, on page 85. It would be easy enough for it to create that TranslatorEnlister inside its awakeFromNib:

```
@translatorEnlister = TranslatorEnlister.alloc.init
```

However, I have a learned aversion to making one class's code explicitly name another class. That tends to make the code harder to change, and it definitely makes it harder to test. Instead, I can make the connection to the TranslatorEnlister be an outlet, no different in principle from the outlets to the button and combo box. That's done at ❶, as shown here:

Download fenestra/reshaped-but-gutted/AppChoiceController.rb

```
class AppChoiceController < Controller

  # Upward to the view
  ib_outlets :comboBox, :button

  # Downward into guts
❶ ib_outlet :translatorEnlister
```

Once we tell it about TranslatorEnlister, nib loading can do the connecting for us. Having the outlet set from outside the class is a convenient form of *dependency injection*.[2]

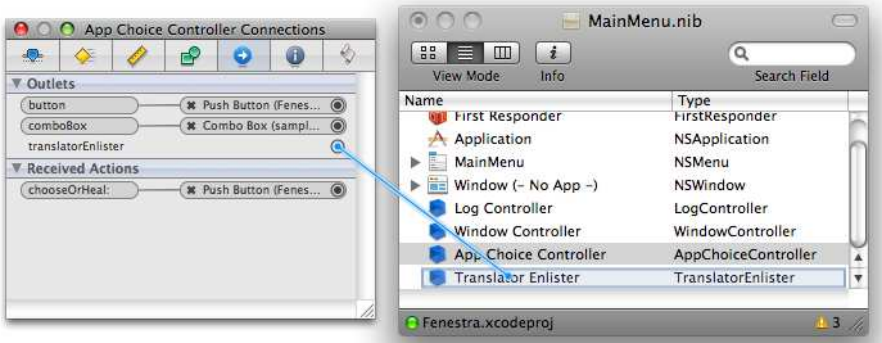
First, we need a TranslatorEnlister to load. In this version of the application, we're just building scaffolding, so there'll be no actual translators. The TranslatorEnlister will just programmatically supply the same two bits of information we already specified in IB: what should go in the combo-box-as-a-list and what should be the initial value of the combo-box-as-a-text-field. Here's a way to do that:

Download fenestra/reshaped-but-gutted/TranslatorEnlister.rb

```
class TranslatorEnlister < OSX::NSObject
  include OSX
  attr_reader :choices, :favorite

  def init
    @favorite = "sample webapp com.exampler.counting"

    @choices = [
      @favorite,
      "for other apps: use.dot.format.name"
    ]
    super_init
  end

  def awakeFromNib
    NSLog("TranslatorEnlister awakes from Nib.")
  end
end
```

To Interface Builder, it doesn't matter in the slightest that TranslatorEnlister has nothing to do with, well, the interface. You create and connect it the same way you would any other object.

2. My favorite article on dependency injection is J. B. Rainsberger's "Injecting testability into your designs" [Rai05].



6.5 Initializing Combo Boxes

Here's how the AppChoiceController can put the information provided by the TranslatorEnlister into the combo box:

Download fenestra/reshaped-but-gutted/AppChoiceController.rb

```
def awakeFromNib
  NSLog("App Choice Controller awakes from Nib.")
❶ @comboBox.removeAllItems
  @translatorEnlister.choices.each do | t |
❷   @comboBox.addItemWithObjectValue(t)
  end
❸ @comboBox.stringValue = @translatorEnlister.favorite
end
```

❶ We've already initialized the list to have two items in Interface Builder. We could remove them there, but it's prudent to clear the list anyway. Remove this line to see a list with duplicates.

❷ This is where items are added to the list. The method name, addItemWithObjectValue, hints that the argument can be something other than a string. Indeed, it can be any object. Try changing the choices array to be an array of integers. You'll see that they display reasonably, and they're correctly logged in chooseOrHeal when the button is clicked.

❸ This line sets the value of the text field. It does nothing to the combo box's list. An alternate way to get the same effect would be to use selectItemAtIndex with the argument 0.

See Apple's *Combo Box Programming Topics* [App08i] for more on combo boxes.

6.6 What Now?

We now have four objects (three controllers and a TranslatorEnlister) that, for the most part, have no references to each other—but they have to exchange information. I'm going to use the notification system (as described in Section 4.1, *Notifications Within an App*, on page 64) to do that. That's reasonably straightforward: every arrow in Figure 6.2, on page 86, turns into the posting of a notification. I'll briefly show what that code looks like in the next chapter. My ulterior motive for doing that is to motivate the chapter after that, which uses Ruby to make such code more pleasant.

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.

Contact Us

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