

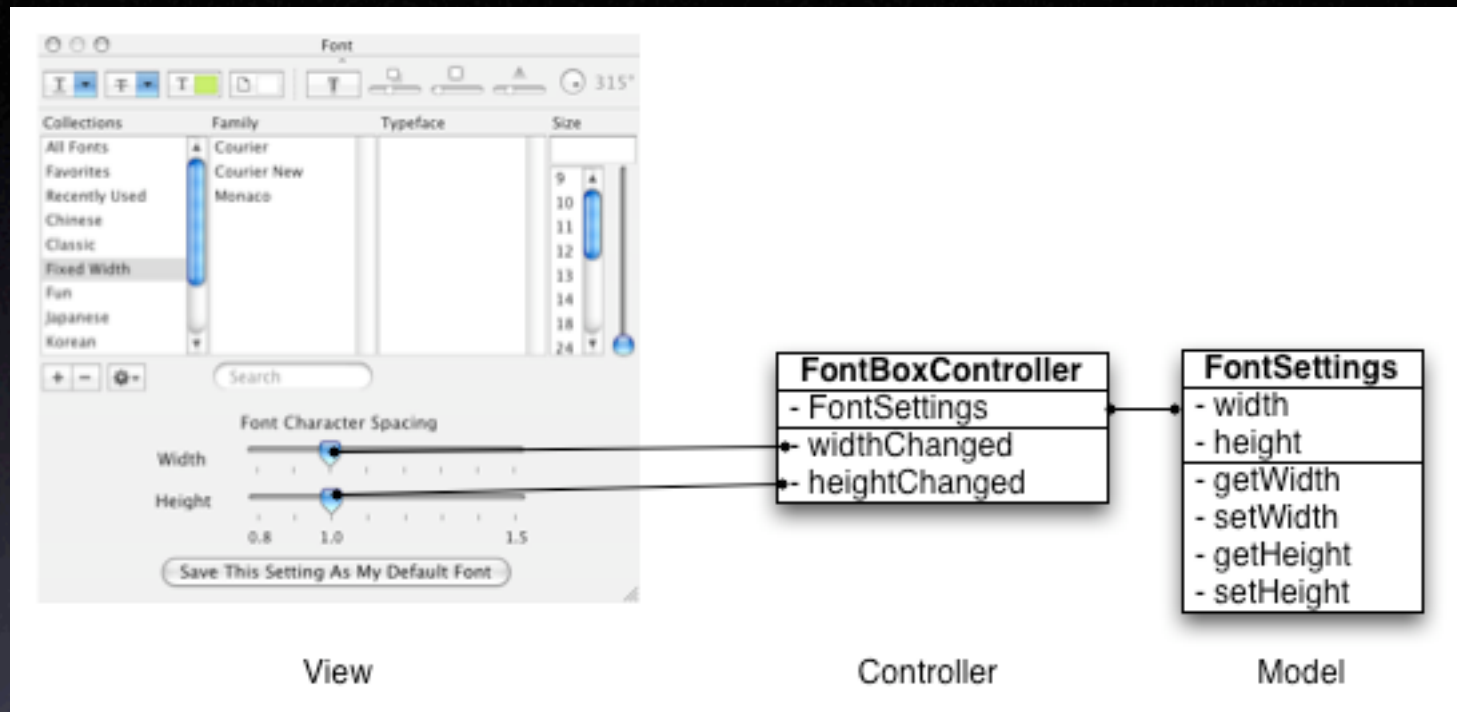
Rapid Application Development with RubyCocoa

Doug Beaver - seattle.rb - 2/24/2005
robotmonk@yahoo.com

What is Cocoa?

- Cocoa is Apple's main application development framework; it is based on NextSTEP technology and has been part of OS X since OS X.0.
- You can write Cocoa applications in ObjC and Java, although ObjC is much more popular.

How Cocoa works



- Cocoa apps use MVC components (model, view, controller) and provide their functionality as callbacks to hooks inside an OS-maintained event loop. GUI features are constructed in Interface Builder, an application that lets you place UI elements on a window and connect callbacks in your code to be run when controls are used.

What is RubyCocoa?

- RubyCocoa is a set of Ruby extensions that let you write Cocoa and AppleScript applications for Apple's OS X. It allows you to implement both controllers and models in Ruby. The views are created with Interface Builder, just like ObjC/Java.
- RubyCocoa can even use your pre-existing NIB files (data files that declare what UI components your windows use, the placement of controls, and their callbacks).

Core Frameworks

- The two major frameworks used for Cocoa apps are AppKit and Foundation.
- Other commonly used frameworks include WebKit (Safari's core internals), QuickTime, Accelerate, OpenGL, ScreenSaver, etc.

AppKit

- AppKit is OS X's programmatic interface to the UI layer. It lets you create windows, controls (buttons, sliders, etc) and hook in rich content (OpenGL, QuickTime movies, images, etc). AppKit is the most common framework called from RubyCocoa.

Foundation

- Foundation is the core "data engine" of OS X applications. It implements network wrappers, heavy strings, notifications, etc. For the most part, your RubyCocoa apps won't be making many calls into the Foundation framework, unless you need to interface with other OS X apps on your computer.

ObjC Bridges

- Programmatic bridges between ObjC and other languages are popular these days. For example, Apple has a JNI bridge that exposes most of its ObjC classes to Java developers on OS X.
- When RubyCocoa is built, it finds all the Apple ObjC headers on your system and dynamically builds the code to wrap them. This pretty much eliminates the need to maintain the bridge code.

Bridge downsides

- Both RubyCocoa and the JNI bridge suffer from poor documentation. Both bridges try to expose the ObjC classes in as idiomatic a form as possible, which has the side-effect of pointing you towards the official Apple docs when you're trying to figure something out. This is not the end of the world, but it does raise the bar for entry when you want to play with RubyCocoa.

Ruby/ObjC Method Transcoding

- `[view addSubview:subview positioned:above relativeTo:otherView];`
- Will be called from Ruby as:
- `view.addSubview(subview, :positioned, above, :relativeTo, otherView);`
- Note that the selector args are fairly well preserved on the Ruby side.

Ruby/ObjC Method Transcoding Cntd

- Other major gotcha is calling ObjC superclasses.
- - (void)drawRect:(NSRect rect) {
 if ([super drawRect]) {}
}
- def drawRect(frame)
 if super_drawRect(frame); end
end

RubyCocoa Capabilities

- Mixed language applications.
- You can create your apps with a mix of ObjC, Java and Ruby. The common case for a mixed app is to use Ruby models and ObjC views and controllers to keep the responsiveness of your application snappy.

RubyCocoa Capabilities

- Runtime loading of ObjC frameworks from your system.
- The bridge lets you create objects using those frameworks and send them methods, so you could load the QuickTime framework and edit movies from Ruby if you wished. If you're making lots of calls to ObjC classes, then it won't be blazing fast, but it will work.

RubyCocoa Setup

- For OS X's default Ruby 1.6.7, you can download a disk image from the RubyCocoa site.
- If you want to use Ruby 1.8.X on Panther, you need to sync from CVS and build manually. This is trivial since SourceForge gives you commands to copy and paste into your shell.

A Word on Performance

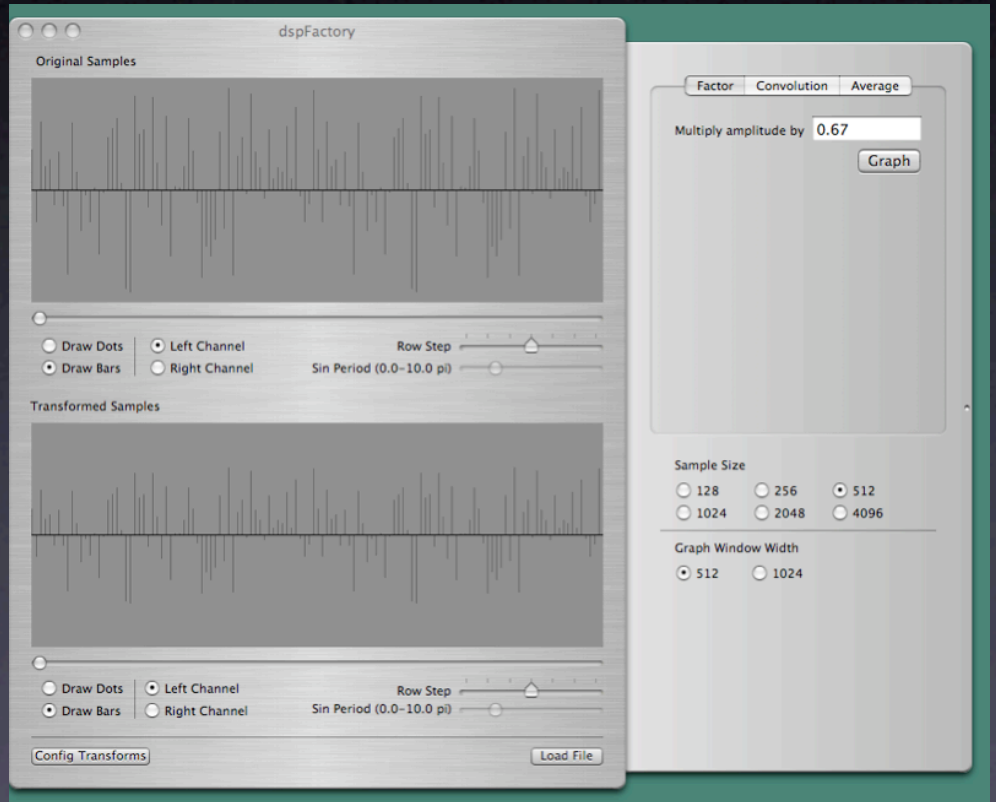
- The bridge is incredibly flexible, but the flexibility comes at a price since method calls are always going through the translation layer and being turned into method calls against the ObjC runtime.
- There are several opportunities for optimization in RubyCocoa, and perhaps RubyInline would be a nice fit for low-hanging fruit.

App Bundles

- You can create bona-fide .app bundles with RubyCocoa, but there's not a way to bundle Ruby itself inside your app yet.
- This is relatively straightforward to do, so I've been working on implementing this. It will allow you to create completely self-contained apps that users can download and use without any external dependencies.

A Case Study

- I have an ObjC application that does crude processing of audio data. It displays samples and lets you apply transformations to the set, graphing what the results look like.



Case Study II

- With my limited RubyCocoa experience, I was able to port most of my app's functionality to RubyCocoa in about eight hours and add a couple new features as well.
- Future prototype apps will only take 2-4 hours as greater familiarity is gained with the RubyCocoa framework.

Why I like RubyCocoa

- I think RubyCocoa needs more polishing before people can deploy applications with it and have users not know that Ruby is under the hood.
- BUT, as it is now, it is a terrific tool for doing rapid prototyping of GUI applications. You can get something working quickly, refactor your class layouts and data model, and then once the dust has settled, port the app to Java or ObjC to get the best performance.

References

- <http://rubycocoa.sourceforge.net>
- <http://developer.apple.com/referencelibrary/Cocoa>
- <http://rubygarden.org/ruby?RubyCocoa>