

Developing iOS Applications With RubyMotion

Matias Korhonen

Bachelor's Thesis
Degree Programme in Business
Information Technology
2013



Degree Programme in Business Information Technology

<p>Author Matias Korhonen</p>	<p>Year of entry 2008</p>
<p>Title of report Developing iOS Applications With RubyMotion</p>	<p>Number of report and attachment pages 50 + 22</p>
<p>Advisor Juhani Välimäki</p>	
<p>Users are spending an ever larger amount of time using smartphones and other mobile devices. Users are more willing than ever to both buy apps for these devices and more willing to buy other products using these devices.</p> <p>Until recently, it has not been possible to use the Ruby language to create applications for any popular smartphone platforms. However, this has relatively recently changed with the introduction of Ruboto for Android and RubyMotion for iOS development.</p> <p>The purpose of this thesis project was to investigate the development of iOS applications in RubyMotion. RubyMotion is a toolchain that enables the development of iOS and OS X applications using the Ruby programming language. RubyMotion applications can use all the same features that are available to regular, Objective-C based applications. Developers can also use 3rd party Objective-C libraries in their RubyMotion applications.</p> <p>The thesis indicated that RubyMotion is a viable option for developers with prior experience in either Ruby or with iOS application development (or both). Nevertheless, at this point, RubyMotion does not seem like a language that should be recommended to novice developers as the mix of Ruby and the iOS SDK can be confusing at times.</p>	
<p>Keywords RubyMotion, Ruby, iOS, mobile, programming, development</p>	

Table of contents

Terms and abbreviations.....	1
1 Introduction.....	3
1.1 Environment and need.....	3
1.1.1 Smartphone market share and consumer behaviour	3
1.1.2 Mobile purchases.....	5
1.1.3 The Ruby programming language.....	6
1.2 Scope.....	7
1.2.1 The scope of this research.....	7
1.2.2 Excluded issues and topics.....	8
2 Research plan.....	9
3 Theory background.....	10
3.1 RubyMotion.....	10
3.1.1 Availability of RubyMotion sources.....	10
3.2 MacRuby.....	11
3.3 The official iOS toolchain.....	11
3.4 How does RubyMotion differ from normal iOS development.....	12
3.5 The benefits of RubyMotion over the official toolchain.....	13
3.6 Research problem.....	13
4 Preparing for development.....	15
4.1 Typical functionality in mobile applications.....	15
4.2 Requirements for the proof of concept application.....	17
4.3 Installing the tools.....	17
4.3.1 Software and hardware used.....	18
4.3.2 Installing Xcode.....	19
4.3.3 Installing RubyMotion.....	21
4.3.4 RubyMotion and text editors.....	21
4.4 Creating a New RubyMotion Application.....	22
4.5 Running The Application.....	22
4.5.1 In a Simulator.....	22
4.5.2 On a Real Device.....	23

5	Development.....	24
5.1	Building the application UI.....	24
5.1.1	The MVC pattern.....	24
5.1.2	Defining layouts programatically.....	25
5.1.3	Using Xcode's Interface Builder.....	27
5.2	Storing data locally.....	28
5.2.1	Core Data.....	29
5.3	On-board sensors.....	30
5.3.1	Device location and map.....	31
5.3.2	Using the camera.....	33
5.4	Accessing a remote API.....	34
5.5	Using 3rd party Objective-C libraries.....	35
5.5.1	CocoaPods.....	35
5.6	Interactive testing.....	36
6	Results, evaluation, and conclusions.....	38
6.1	Results.....	38
6.2	Evaluation.....	38
6.2.1	Building the application user interface.....	39
6.2.2	Storing data locally.....	39
6.2.3	On-board sensors.....	39
6.2.4	Accessing a remote API.....	40
6.2.5	Using 3 rd party Objective-C libraries (via CocoaPods).....	40
6.3	Conclusions.....	40
7	Summary.....	42
7.1	Further research questions.....	43
	References.....	44
	Attachments.....	48

Terms and abbreviations

API	Application Programming Interface. An API specifies how software components can interact with each other.
CLI	Command-Line Interface. A textual interface used for interacting with computer programs.
CocoaPods	A dependency manager for Objective-C projects and libraries
GPS	Global Positioning System. The primary system used to acquire the position of mobile devices.
HTTP	Hyper Text Transfer Protocol. The application protocol used for most services on the internet.
JSON	JavaScript Object Notation. A common data format derived from the JavaScript programming language
LLVM	Formerly for “Low Level Virtual Machine.” A language agnostic compiler infrastructure that can be used to create compilers for programming languages.
MVC	Model-View-Controller. A common programming design pattern for increased code modularity and reusability.
Objective-C	An object-oriented programming language based on the C programming language.
REST	Representational State Transfer. An architectural style commonly used for web services.

REPL	Read-Eval-Print-Loop. The interactive shell that ships with RubyMotion for interactive debugging.
Ruby	A dynamic, object-oriented programming language.
RubyMotion	An implementation of the Ruby programming language that runs on the iOS and OS X platforms.
Terminal	The default terminal emulator application on OS X.
Toolchain	A set of programming tools used to create applications
USB	Universal Serial Bus. The most common interface used for connecting external devices to computers.
WYSIWYG	What You See Is What You Get. A system in which content displayed in an editor resembles closely what the finished product will look like.
Xcode	Apple's IDE for iOS and OS X software development
XML	Extensible Markup Language. A data format commonly used for web services.

1 Introduction

RubyMotion is an Toolchain built on top of MacRuby what enabled iOS and OS X application development using the Ruby programming language. RubyMotion compiles Ruby to native machine code using LLVM. (HipByte, 2013)

“RubyMotion implements Ruby on top of the Objective-C runtime and Foundation classes” (HipByte, 2013). This means that RubyMotion applications have access to all native iOS and OS X APIs and can also use 3rd party Objective-C libraries.

1.1 Environment and need

Both the amount of time and the amount of money spent on mobile devises is on the rise. This present an opportunity to developers willing to invest time in learning these platforms.

1.1.1 Smartphone market share and consumer behaviour

Android phones dominate the smartphone market: 81 percent of devices shipped with Android in the 3rd quarter of 2013. Only 12.9% of devices shipped with Apple's iOS platform. (Dilger, 2013)

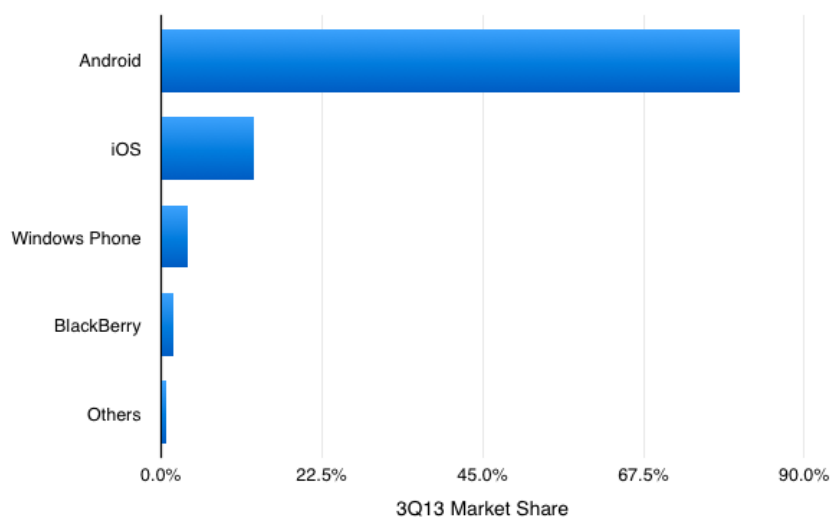


Figure 1: The Android operating system dominates the number of devices shipped. (Dilger, 2013)

At first glance those figures sound dismal for Apple and for iOS application developers, but Apple makes 56% of all profits in the smartphone device market (Bradley, 2013) and Apple users spend more money on applications than Android users (OPA, 2012, p. 30).

Apps users who go to the Apple App Store tend to download nearly twice as many apps as those who go to the Android Market or the BlackBerry App World Store. They also seem more willing to pay for their apps: Apple App Store customers report that for every two free apps they download, they typically pay for one. In contrast, apps users who frequent the Android Market and BlackBerry App World stores report downloading more than 3.5 free apps for every one they buy.

(Nielsen, 2010, p. 5)

As you can see from Figure 2 and from the quote above, iPhone users are more likely to pay for apps in *every* price category. For iOS application developers this means that consumers on the iOS platform are more likely to buy their app.

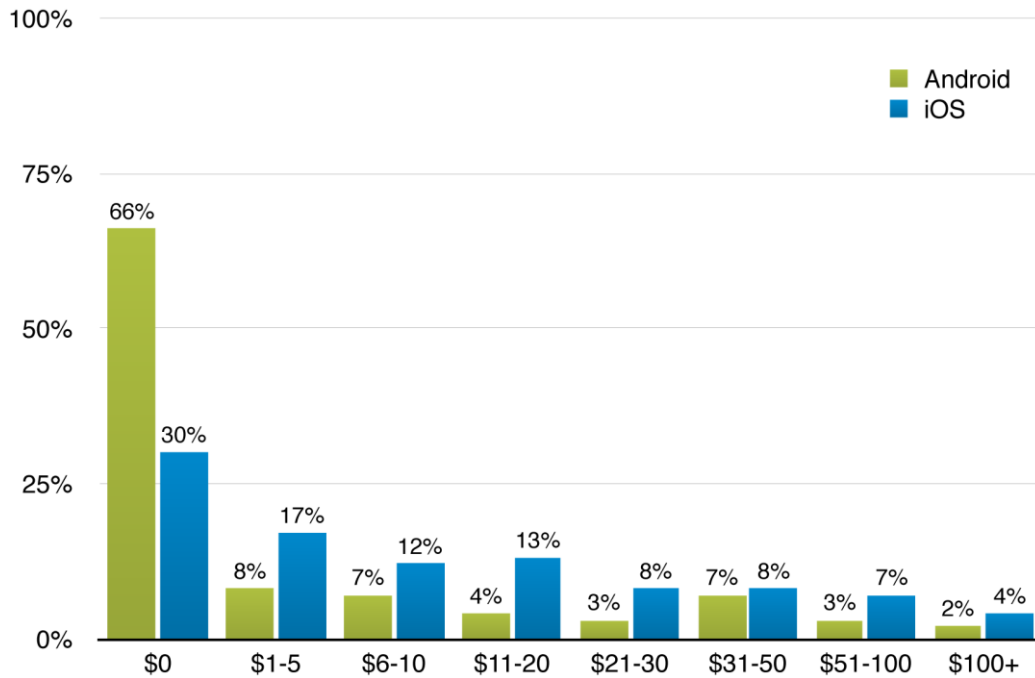


Figure 2. Amount of Money Spent on Smartphone Apps in the Last Year (% of Smartphone Content Consumers) (OPA, 2012, p. 30)

1.1.2 Mobile purchases

Consumer spending and purchases using mobile phones and tablets is on the increase. According to eMarketer, 47% of all mobile phone owners in Western Europe will own a smartphone by the end of 2013. “12.4% of all seasonal online sales in key Western European markets will be made on mobile devices this year. Mobile sales will be 68% higher than during the holiday period in 2012.” (eMarketer, 2013)

According to data gathered by comScore, 14.6% of smartphone owners in Europe make online purchases online. The most commonly purchased items are clothing, consumer electronics or appliances, and books. (comScore, 2013)

1.1.3 The Ruby programming language

Rank	RedMonk	TIOBE	The Language Popularity Index
1	JavaScript	C	C
2	Java	Java	Java
3	PHP	Objective-C	Objective-C
4	Python	C++	C++
5	Ruby	C#	Basic
6	C#	PHP	PHP
7	C++	Visual Basic	Python
8	C	Python	C#
9	Objective-C	Transact-SQL	Perl
10	Perl	JavaScript	Ruby
11	Shell	Visual Basic .NET	Pascal
12	Scala	Perl	JavaScript
13	ASP	Ruby	Ada
14	Haskell	Pascal	R
15	Assembly	Lisp	Lisp/Scheme

Figure 3: Relative ranking of the top 15 programming languages on the web from three programming language ranking sources. Ruby is highlighted. (RedMonk, 2013) (The Transparent Language Popularity Index, 2013) (TIOBE Software, 2013)

Depending on which statistics one looks at, Ruby ranks between the 5th and 13th most popular (see Figure 3yllä).

In the TIOBE Programming Community Index, which tracks programming language popularity on the web, Ruby ranked 13th in popularity in November 2013, down from 10th in November 2012 (TIOBE Software, 2013).

The RedMonk Programming Language Rankings, which derives its rankings from GitHub and Stack Overflow, places Ruby as high as fifth. (RedMonk, 2013)

Despite this relative popularity, historically it has not been possible to develop smartphone application using Ruby. This changed with the introduction of Ruboto in 2009 (Nutter, 2009) and the release of RubyMotion for iOS in 2012 (Paul, 2012).

1.2 Scope

Overall the purpose of this research is to evaluate the prospect of using the RubyMotion toolchain for iOS application development. The purpose is not to produce a genuinely useful application for real users.

1.2.1 The scope of this research

While it is possible to use RubyMotion 2.0 to develop both iOS and OS X applications (Suri, 2013), the scope of this research is limited to only iOS applications.

The goal of this research is simply to assess whether the development of iOS applications using the RubyMotion toolchain is viable. To this end the proof of concept application will include typical aspects of mobile applications, for example presenting data from APIs to the user and letting the user query or filter data from these APIs.

The application itself will only be for technical testing of common mobile application features and not for a specific need. These features might include some all or all the following features:

- Remote JSON or XML API access (via HTTP)
- Presenting data from the above APIs to the user in a useable form (for example tables, charts, or icons)
- Integrating features from 3rd party libraries into the application

As the application is only intended to be used for technical testing, there are no precise requirements. The success of the proof of concept application will be evaluated by hand and by whether the accomplishes the goals set in the reasearch plan. It is possible that this application will be split into a series of smaller applications if necessary.

1.2.2 Excluded issues and topics

Developing OS X applications with RubyMotion is not within the purview of this research. At the same time intensive performance benchmarking is also not included at this time.

While performace testing is out of the scope of this paper, the proofing and evaluation of RubyMotion will be done by building an application that represents the typical requirements of an iOS application. The performance of this application will be done by look and feel, for example:

- Does it feel like a normal iOS application?
- Does it feel responsive or slow?
- Does it start up fast?
- Can you tell that the application wasn't built with the official tools?

2 Research plan

A proof of concept or proof of technology application will be developed to evaluate how effective the RubyMotion toolchain is for typical aspects of mobile applications. A secondary goal of the development process will be assess how pleasant or easy the development of iOS applications is with RubyMotion.

These aspects can include features such as presenting data from remote HTTP APIs to the user and letting the user query or filter data from these APIs. The application will also be used to test how native 3rd party Objective-C libraries can be used in conjunction with the RubyMotion application.

The purpose of this application is not to be a useful or viable application on it's own, but to test how or if RubyMotion can be used to develop the selected features.

Proof-of-Concept Prototype is a term that (I believe) I coined in 1984. It was used to designate a circuit constructed along lines similar to an engineering prototype, but one in which the intent was only to demonstrate the feasibility of a new circuit and/or a fabrication technique, and was not intended to be an early version of a production design
(Carsten, 1989)

By developing the selected features, we will be able to form a rough picture of how RubyMotion can be used and whether it fulfils its promise. It will also be possible to evaluate how difficult it is for a Ruby developer to begin to do iOS development using the RubyMotion tools.

3 Theory background

3.1 RubyMotion

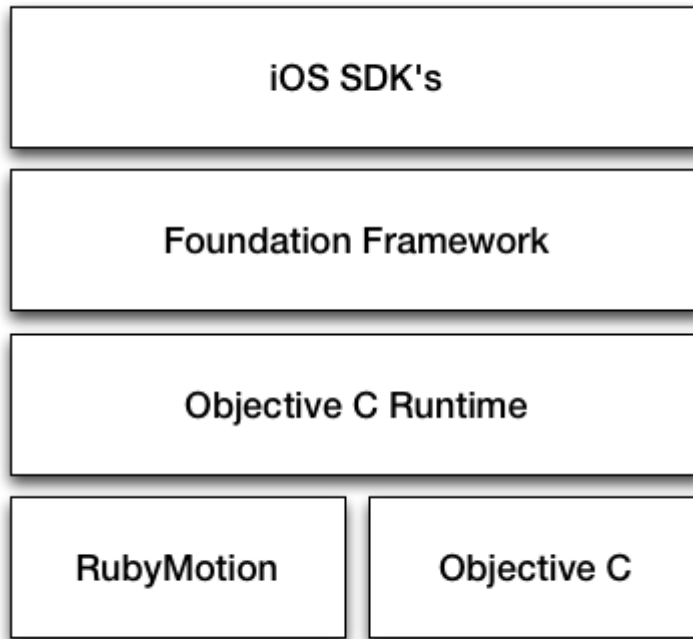


Figure 4: "RubyMotion is a toolchain that allows developers to develop native iOS applications using the Ruby programming language. RubyMotion acts as a compiler that interacts with the iOS SDK" (Nalwaya & Paul, 2013, p. 10)

RubyMotion was created by Laurent Sansonetti, the creator of MacRuby. RubyMotion itself is based on the open-source MacRuby project, but implements a proprietary LLVM compiler that converts Ruby code to native machine code. (Nalwaya & Paul, 2013, p. 10)

3.1.1 Availability of RubyMotion sources

RubyMotion is a relatively new toolchain, having only been introduced in late 2012 (Paul, 2012). This newness translates into a relative lack of reliable independent and third party sources. In particular no reliable sources benchmarking RubyMotion

performance on iOS were found nor any direct, quantitative performance comparisons between application built in Objective-C and applications built with RubyMotion.

3.2 MacRuby

RubyMotion was born out of the open source MacRuby project. MacRuby provides a seamless bridge between Ruby code and the Cocoa eco system on OS X. While RubyMotion is based on the open source MacRuby project, RubyMotion itself is not open source and it uses an LLVM-based compiler to convert Ruby code into machine code. (Paul, 2012)

According to HipByte, the company behind RubyMotion, this means that RubyMotion applications are as fast (or in some cases faster) than applications written in Objective-C. (HipByte, 2013)

Laurent Samsonetti announced the first version of MacRuby in 2008 on the Ruby-Talk mailing list. (Samsonetti, [ANN] MacRuby 0.1, 2008)

MacRuby is an implementation of Ruby 1.9 directly on top of Mac OS X core technologies such as the Objective-C runtime and garbage collector, the LLVM compiler infrastructure and the Foundation and ICU frameworks. It is the goal of MacRuby to enable the creation of full-fledged Mac OS X applications which do not sacrifice performance in order to enjoy the benefits of using Ruby.

(The MacRuby Team, 2013)

3.3 The official iOS toolchain

The official tools endorsed and supplied by Apple for iOS application development are the Xcode IDE and the Objective-C programming language. (Apple Inc., 2013a)

3.4 How does RubyMotion differ from normal iOS development

RubyMotion lets the developer use all the normal iOS SDKs just like an Objective-C developer would, as in the example in Figure 5. (Symonds, 2012)

<p>Objective-C example:</p> <pre>UITableViewCell *cell = [self.tableView dequeueReusableCellWithIdentifier:CellID]; if(cell == nil) { cell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault reuseIdentifier:MyCellIdentifier]; }</pre>
<p>Equivalent in RubyMotion:</p> <pre>cell = tableView.dequeueReusableCellWithIdentifier(CellID) UITableViewCell.alloc.initWithStyle(UITableViewCellStyleSubtitle, reuseIdentifier:CellID)</pre>

Figure 5: Creating a table view cell in Objective-C and its equivalent with RubyMotion. (Symonds, 2012)

Alternatively, developers can use wrappers written in Ruby in order to provide a more pleasant coding experience (Figure 6). This means that less code needs to be written per application and the code looks more intuitive. (Allsopp, The RubyMotion Way, 2012b)

<p>Objective-C example:</p> <pre>[button addTarget:self action:@selector(buttonTapped:) forControlEvents:UIControlEventTouchUpInside]; // Elsewhere - (void)buttonTapped:(id)sender { self.view.backgroundColor = [UIColor redColor]; }</pre>
<p>Equivalent in RubyMotion (using the BubbleWrap library):</p> <pre>button.when(UIControlEventTouchUpInside) do self.view.backgroundColor = UIColor.redColor end</pre>

Figure 6: An example of code in Objective-C and its equivalent in RubyMotion. The RubyMotion example uses the BubbleWrap library. (Allsopp, The RubyMotion Way, 2012b)

3.5 The benefits of RubyMotion over the official toolchain

Developers who are already familiar with the Ruby programming language have the most to gain from RubyMotion (if it is found to be a viable tool for iOS application development) as they won't have to learn a whole new programming language in order to develop applications for the iOS platform.

Objective-C can also be quite a verbose language, so it is possible that using RubyMotion for iOS application development could lead to shorter development times.

RubyMotion also provides access to an interactive debugging shell, called the REPL (for read-print-evaluate-loop). This allows the developer to modify an application that is running in the simulator without needing to restart it. The interactive shell lets the developer quickly and simply debug views or experiment with how they could be changed. (Samsonetti, RubyMotion Project Management Guide, 2013b)

3.6 Research problem

The goal of this research is to investigate whether building iOS applications using Ruby and RubyMotion is viable, especially for developers with no previous experience with either Objective-C or the iOS APIs.

The application or applications developed during the course of this research should attempt to recreate some of the most common features of typical smartphone applications.

These features should be used to assess whether using RubyMotion for iOS application development is viable, particularly for existing Ruby developers (rather than, say, programming novices or Objective-C developers).

4 Preparing for development

4.1 Typical functionality in mobile applications

By looking at the tables of contents for six books on mobile application development (on iOS, Android, and Windows Phone), the following table of common features was drawn up based on the chapter titles.

The books used to devise this table (Figure 7) were (or rather their tables of content):

- Hello Android
- Programming Windows Phone 7
- Programming Android
- Learning iOS Programming
- RubyMotion: iOS Development with Ruby
- RubyMotion iOS Development Essentials

These books were selected because they represent a variety of authors and publishers, and thus are likely to represent the typical needs of typical smartphone applications.

	Hello Android	Programming Windows Phone 7	Programming Android	Learning iOS Programming	RubyMotion: iOS Development with Ruby	RubyMotion iOS Development Essentials
Installing the tools	Yes	No	Yes	Yes	Yes	Yes
Running on an emulator/simulator	Yes	No	Yes	Yes	Yes	Yes
Running on a real device	Yes	No	Yes	Yes	Yes	Yes
Designing and building the UI	Yes	Yes	Yes	Yes	Yes	Yes
2D graphics	Yes	Yes	Yes	No	Yes	Yes
3D graphics	Yes	Yes	Yes	No	No	No
Multimedia (audio and video playback)	Yes	No	Yes	No	No	No
Storing local data	Yes	Yes	Yes	Yes	Yes	Yes
Using webviews	Yes	No	No	Yes	No	Yes
Location and other sensors	Yes	Yes	Yes	Yes	No	Yes
Multi-touch	Yes	Yes	No	No	No	Yes
Application testing	Yes	No	No	No	Yes	Yes
Distributing the application	No	No	Yes	Yes	Yes	Yes
Accessing remotes APIs	Yes	No	Yes	Yes	Yes	Yes

Figure 7: Common smartphone application features based on mobile programming books. (Burnette, 2010, pp. vi-ix) (Petzold, 2010, pp. iv-xii) (Dornin, Mednieks, Meike, & Nakamura, 2012, pp. iii-xi) (Allan, 2013, pp. iii-viii) (Allsopp, *RubyMotion: iOS Development with Ruby*, 2012a, pp. v-vi) (Nahaya & Paul, 2013, pp. i-v)

4.2 Requirements for the proof of concept application

Based on the table of typical mobile application features, the following list of features was selected for the proof of concept application:

- Installing the tools
- Running the application on a simulator
- Running the application on a real device
- Designing and building a simple UI for the application
- Storing data locally
- Getting the device location from the on-board sensors (GPS)
- Getting data from a remote API

In addition to these topics, the proof of concept application will also test how a 3rd party Objective-C library can be integrated into the RubyMotion application.

The development of these features will be used to evaluate whether RubyMotion is a viable toolchain for developing applications for the iOS platform.

4.3 Installing the tools

Using RubyMotion requires the developer to install both Xcode and the RubyMotion tools. As RubyMotion requires Xcode to be installed RubyMotion applications cannot be developed on non-Apple hardware or operating systems. (Samsonetti, Welcome to RubyMotion, 2013a)

4.3.1 Software and hardware used

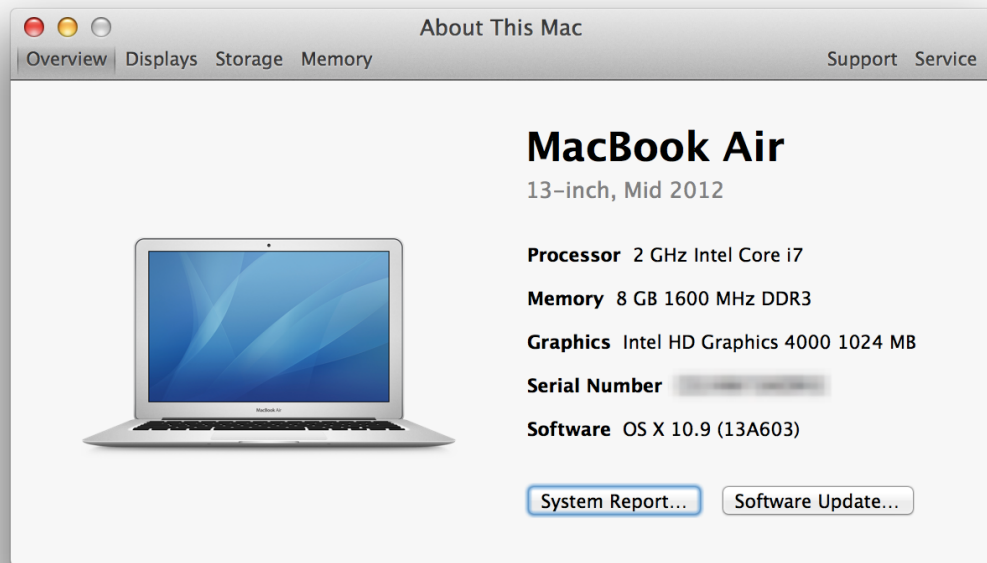


Figure 8: Hardware and operating system details of the development machine. The serial number has been blurred.

These installation procedures were tested and checked on a mid 2012 Apple MacBook Air running the latest version of OS X available at the time, OS X 10.9 Mavericks. RubyMotion v2.16, the latest available version of RubyMotion, was used.



Figure 9: Hardware and iOS version details of the iPhone that was used for testing. The serial number and other identifiers and been blurred.

The apps were tested on iOS 7.0.4, the latest version available, on the iOS Simulator and on an iPhone 5 running the same version of iOS.

4.3.2 Installing Xcode

Installing Xcode itself is extremely simple, the developer simply needs to open the “App Store” application on their development machine and search for “Xcode” to locate the Xcode application in the store. Clicking “Install” will install the Xcode development environment and the latest iOS SDK (a free Apple ID account is required for the installation to proceed).



Figure 10: Xcode on the Mac App Store. In this screenshot Xcode has already been installed.

Once Xcode itself has been installed, the developer needs to install the command line tools. In previous version of Xcode and OS X (prior to Xcode 5.0.1 and OS X 10.9) the command line tools could be install from within Xcode, but this has now changed (Zaph, 2013).

The official RubyMotion “Getting Started” tutorial has not been updated to reflect this change at the time of writing. (Samsonetti, Welcome to RubyMotion, 2013a)

With the latest versions of OS X and Xcode, the command line tools need to be installed by executing a single command (`xcode-select --install`) on the CLI (Command Line Interface). (Zaph, 2013)

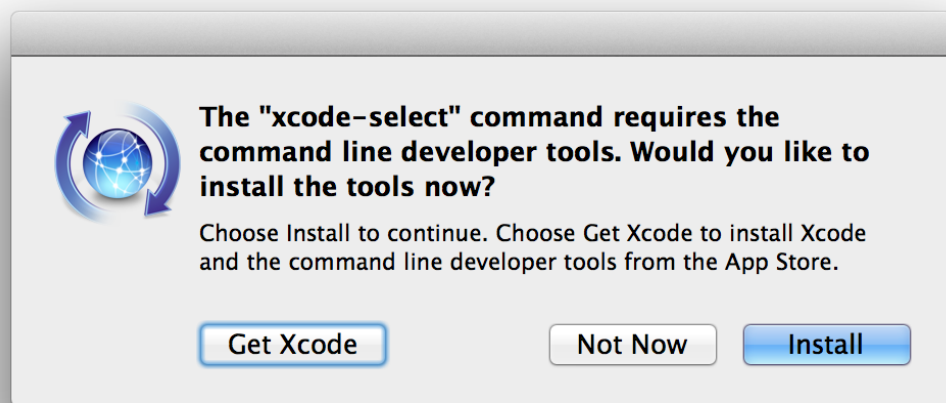


Figure 11: Running the "xcode-select --install" command results in this dialog. After clicking "Install", the command line tools will be downloaded and installed.

4.3.3 Installing RubyMotion



Figure 12: The RubyMotion installation wizard.

Installing the RubyMotion tools is also very simple (after Xcode and the Xcode Command Line Tools have been installed). After downloading the RubyMotion package from the URL given in the RubyMotion license instructions, all that is required is for the developer to follow the instructions given on screen. After a few minutes, the `motion` command will be available for use in Terminal.

4.3.4 RubyMotion and text editors

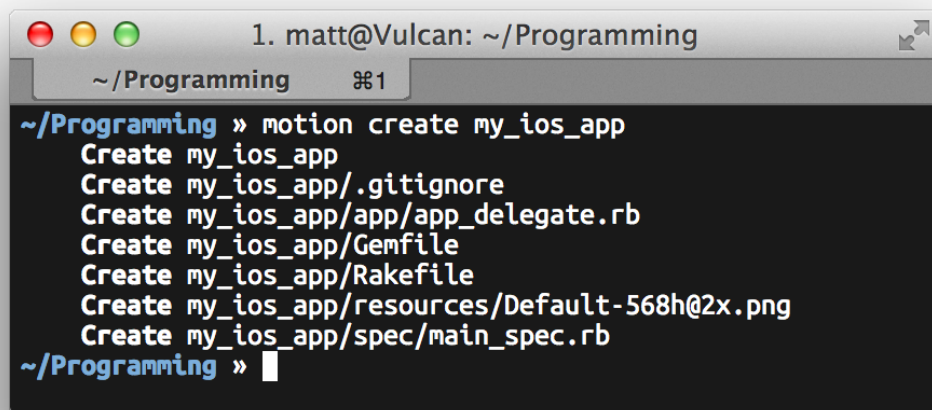
While normally iOS application code is developed and written within Xcode, this is not a requirement for iOS application development. A developer using RubyMotion is free to choose any text editor or IDE they wish. (Nalwaya & Paul, 2013, p. 18)

4.4 Creating a New RubyMotion Application

Creating a new RubyMotion application starts with initializing a new application using the RubyMotion command line tool, `motion`:

```
motion create my_application
```

Running this command will generate an application skeleton:

A screenshot of a macOS terminal window. The window title is "1. matt@Vulcan: ~/Programming". The terminal shows the command "motion create my_ios_app" being executed. The output lists several files and directories created: "my_ios_app", ".gitignore", "app/app_delegate.rb", "Gemfile", "Rakefile", "resources/Default-568h@2x.png", and "spec/main_spec.rb". The prompt returns to "~/Programming »".

```
~/Programming » motion create my_ios_app
Create my_ios_app
Create my_ios_app/.gitignore
Create my_ios_app/app/app_delegate.rb
Create my_ios_app/Gemfile
Create my_ios_app/Rakefile
Create my_ios_app/resources/Default-568h@2x.png
Create my_ios_app/spec/main_spec.rb
~/Programming »
```

Figure 13: The application initialization command displays what files it created for the developer.

4.5 Running The Application

The RubyMotion command line tool can run the application in the iOS simulator (which is included with Xcode) or on a real iOS device that is connected to the development machine, for example an iPhone or iPad.

4.5.1 In a Simulator

The newly created application can be run in the iOS simulator by running the `rake` command within the project folder in terminal:



*Figure 14: *By default the RubyMotion doesn't have any views to display, but it can still be run within the simulator.*

The **rake** command automatically build the application and launches it within the iOS simulator.

4.5.2 On a Real Device

The application can be run on a real, physical iOS device (e.g. an iPhone) by running the **rake device** command while the device is connected to the development machine over USB. (Samsonetti, RubyMotion Project Management Guide, 2013b)

Just as with iOS development using apple's official tools, this requires that the device has been registered on Apple's Development portal (<https://developer.apple.com>) and that the developer has a valid iOS Developer Program subscription. An iOS Developer Program subscription is also required for application distribution via the iOS App Store. (Nalwaya & Paul, 2013, pp. 19,213)

5 Development

5.1 Building the application UI

The iOS SDK provides a few different ways in which the application UI can be built and all of them can be used with RubyMotion.

5.1.1 The MVC pattern

iOS application typically use a design pattern or programming paradigm known as MVC (Model-View-Controller). Models represent data, views display that data to users, and controllers handle user input and act as an intermediary between views and models. (Nalwaya & Paul, 2013, pp. 55-57)

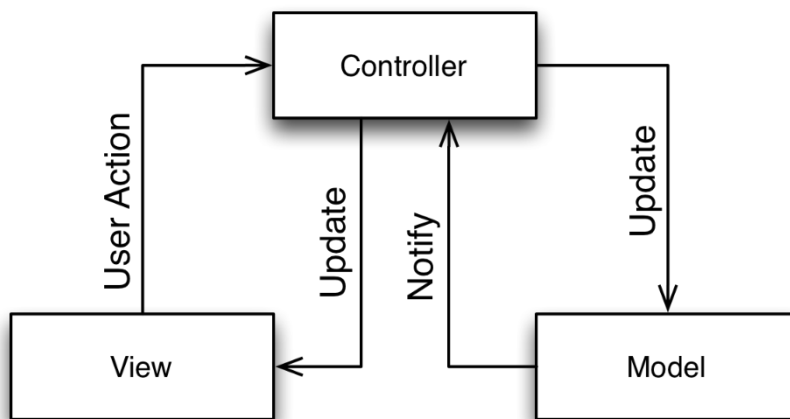


Figure 15: The workings of the MVC model illustrated. (Nalwaya & Paul, 2013, p. 57)

The **View** and **Controller** layers interact through **UserAction** and **Update** as shown in the diagram. Whenever the **View** layer creates or modifies data, it is communicated to **Controller** through **User Action**. Similarly, whenever **Model** updates any change it will first **Notify** the **Controller** and will then be reflected on the **View** by an **Update**

(Nalwaya & Paul, 2013, p. 57)

5.1.2 Defining layouts programmatically

The normal way of defining views when using RubyMotion is to define them programmatically. Defining views programmatically means that Xcode's graphical, WYSIWYG (What You See Is What You Get) Interface Builder (also referred to as IB) is not used, instead the views are defined in code.

Hard-coded interface element positions

The simplest (if most verbose) way of positioning user interface elements onto the view is to add the elements to the view in code and specify their positions at the time of coding. The downside of this is that the views can become very difficult to visualize and making changes is labourious. (Allsopp, RubyMotion: iOS Development with Ruby, 2012a, p. 80)

For the programmatic views the basic “Color” application was implemented from “RubyMotion: iOS Development with Ruby”, with a few minor changes.

```
class ColorsController < UIViewController
  def viewDidLoad
    super

    self.view.backgroundColor = UIColor.whiteColor

    @label = UILabel.alloc.initWithFrame(CGRectZero)
    @label.text = "Colors"
    @label.sizeToFit
    @label.center =
      [self.view.frame.size.width / 2,
       self.view.frame.size.height / 2]
    @label.autoresizingMask =
      UIViewAutoresizingFlexibleBottomMargin | UIViewAutoresizingFlexibleTopMargin
    self.view.addSubview(@label)
  end
end
```

Figure 16: An example of defining a view programmatically in a UIViewController. The positions of view elements are defined in the application code. (Allsopp, RubyMotion: iOS Development with Ruby, 2012a, p. 28)

This method of defining application user interfaces worked just as documented, without any difficulties. While this method of implementing views worked without a hitch, it can get laborious and tedious with more complicated layouts, especially if multiple screen sizes and screen orientations (portrait and landscape) need to be taken into account.

Auto Layout

Auto Layout is a relatively new approach that can be used to define iOS application layouts. It has been available for developers since iOS 6. With auto layout the developer defines layout **constraints** for the various UI elements. The constraints define the relationships between different elements in the interface, e.g. “these buttons should be side-by-side.” (Armando, 2012)

```
Motion::Layout.new do |layout|
  layout.view self.view
  layout.subviews subviews
  layout.metrics "top" => 150, "margin" => 20, "height" => 40
  layout.vertical(
    "|-top-[label(==height)]-margin-[red_button(==height)]-margin-[green_button(==height)]-margin-[blue_button]|")
  layout.horizontal "|-margin-[label]-margin-|"
  layout.horizontal "|-margin-[red_button]-margin-|"
  layout.horizontal "|-margin-[green_button]-margin-|"
  layout.horizontal "|-margin-[blue_button]-margin-|"
end
```

Figure 17: An example of defining the positions of interface elements using the Visual Format Language. In this case the motion-layout library¹ is being used to provide a more Ruby-like interface to the iOS Auto Layout SDK.

For the Auto Layout based experiment, the code from the “Color” app developed using the “Hard coded” method was taken and an attempt was made to convert it to use Auto Layout.

¹ <https://github.com/qrush/motion-layout>

Based on the available documentation, at no point did the Auto Layout based interface work in a satisfactory way. While very basic layouts worked okay, anything with more than two or three subviews would give cryptic errors and did not work as it should have.

The difficulties using autolayout can probably be attributed to the poor level of documentation available. In theory auto layout should provide a more pleasant user interface development process, but in practice this was not the case.

5.1.3 Using Xcode's Interface Builder

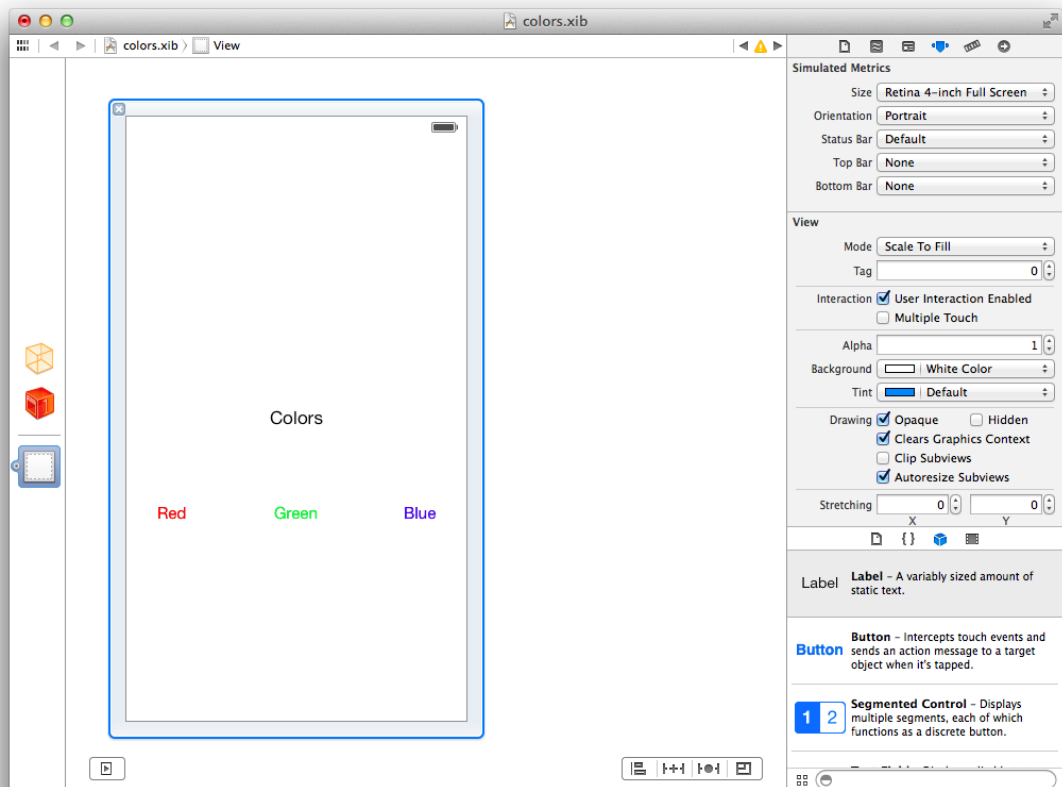


Figure 18: The Xcode Interface Builder

Xcode Interface Builder is a graphical tool used to create iOS and OS X application interfaces. Since version 1.3, RubyMotion has had built in support for using `.xib` files

from Xcode's Interface Builder to define UIs in applications built with RubyMotion. (Phillips, 2012)

For the Interface Builder based experiment, the “Color” example application was once again used. Building the view in Interface Builder was a very pleasant experience and relatively intuitive due to the tools's WYSIWYG nature. Getting the RubyMotion application to use the UI elements created in the Interface Builder was a quick and intuitive experience.

```
def viewDidLoad
  super

  self.title = "Colors"

  @label = view.viewWithTag 1
  red_button = view.viewWithTag 2
  green_button = view.viewWithTag 3
  blue_button = view.viewWithTag 4

  red_button.addTarget(self,
    action: 'tap_red',
    forControlEvents: UIControlEventTouchUpInside)
  green_button.addTarget(self,
    action: 'tap_green',
    forControlEvents: UIControlEventTouchUpInside)
  blue_button.addTarget(self,
    action: 'tap_blue',
    forControlEvents: UIControlEventTouchUpInside)
end
```

Figure 19: Wiring up the user interface created in Inter Face builder to actions in a UIViewController

As Interface builder provided visual tools for configuring Auto Layout, Auto Layout was also used for this example. In this case using Auto Layout provided no additional challenges and the experience stood in stark contrast to the attempt to use Auto Layout programmatically.

5.2 Storing data locally

iOS provides a few different places where user data can be stored by applications: Core Data, XML files, SQLite, and the Keychain. All of these data stores can be used by

developers, though they are intended for different use cases (Apple Inc., 2013b) (Apple Inc., 2013c)

The keychain is intended to be used for storing very small amounts of sensitive data, such as passwords, API tokens, and other secrets. (Apple Inc., 2013c)

XML files can be used to store user settings and preferences in a lightweight format and the iOS SDK provides an API for serializing objects. (Apple Inc., 2013c)

iOS applications can also use SQLite directly. Unlike with Core Data, using SQLite the developer can make procedural, SQL-based queries. (Apple Inc., 2013b)

The primary store for user data is Core Data, which is built on top of SQLite but provides abstractions so to make it work seamlessly with iOS's MVC architecture. (Apple Inc., 2013b)

This section of development focuses on Core Data.

5.2.1 Core Data

The Core Data framework provides comprehensive and automated solutions related to an object's life cycle and its searching and persistence features. It can retrieve and manipulate data purely on an object level without having to worry about the details of storage and retrieval. (Nalwaya & Paul, 2013, p. 130)

To test storing user specific data locally, the Core Data example application from “RubyMotion essentials” was implemented with some minor differences.

By following the instructions given, the local data storage was relatively easy to implement, though rather verbose. The verbosity of the code required is largely due to the iOS Cocoa SDK API design.

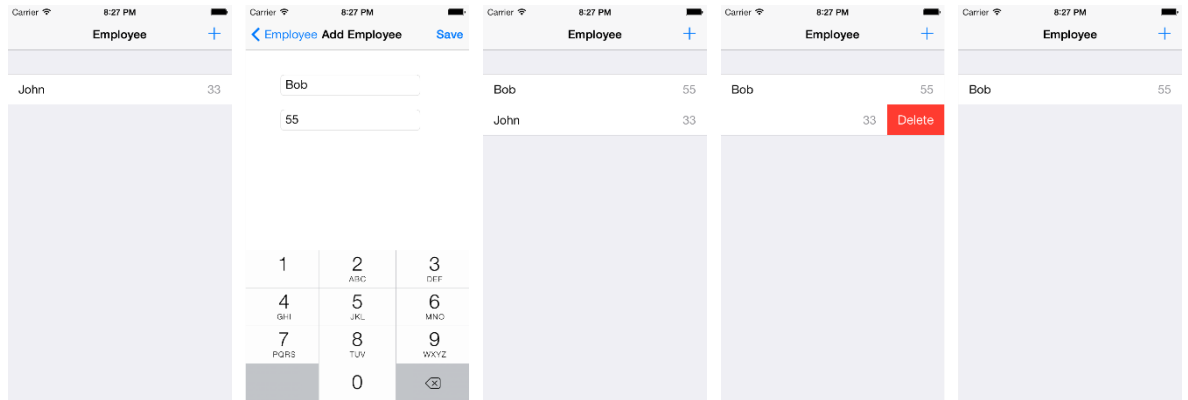


Figure 20: User flow in the example application from left to right: Initial view, adding a new entry, list view with new entry shown, deleting an entry, and finally the list view after deletion.

5.3 On-board sensors

Modern iPhones have a variety on-board sensors available for use, for example the GPS, camera, accelerometer, and compass. Developers can use all of these via the iOS SDKs.

5.3.1 Device location and map

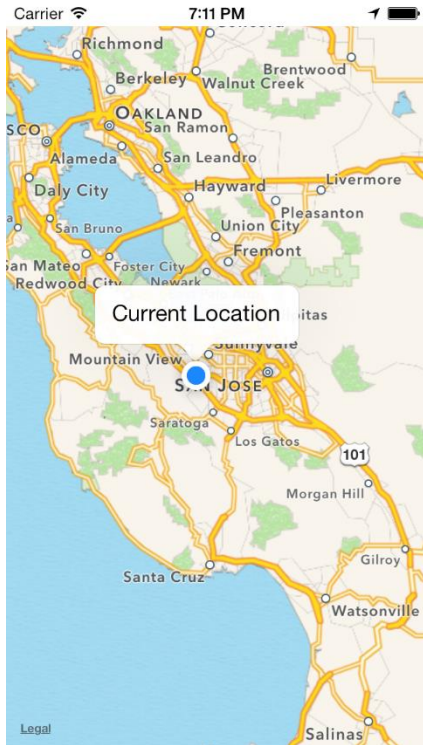


Figure 21: The current device location (simulated to be in California in this example)

For the location test, the a simple application was created to fetch the device's current location and display it on a map. The application used the native iOS SDKs for both fetching the device location and for displaying the map.

```
class LocationController < UIViewController
  def viewDidLoad
    view.backgroundColor = UIColor.whiteColor
    self.title = "Location"

    check_location
    show_map
  end

  def check_location
    if (CLLocationManager.locationServicesEnabled)
      @location_manager = CLLocationManager.alloc.init
      @location_manager.desiredAccuracy = KCLLocationAccuracyKilometer
      @location_manager.delegate = self
      @location_manager.purpose = "This app's functionality is based on your location"
      @location_manager.startUpdatingLocation
    else
      show_error_message('Enable the Location Services for this app in Settings.')
    end
  end
end
```

```

end

def show_map
  map = MKMapView.alloc.initWithFrame( [[0, 20], [320, 568]] )
  map.mapType = MKMapTypeStandard
  map.showsUserLocation = true

  location = CLLocationCoordinate2D.new(@latitude.to_f, @longitude.to_f)
  map.setRegion(
    MKCoordinateRegionMake(location, MKCoordinateSpanMake(1, 1)),
    animated:true
  )

  self.view.addSubview(map)
end

def locationManager(manager, didUpdateToLocation:newLocation, fromLocation:oldLocation)
  @latitude = newLocation.coordinate.latitude
  @longitude = newLocation.coordinate.longitude

  @location_manager.stopUpdatingLocation

  show_map
end

def locationManager(manager, didFailWithError:error)
  show_error_message('Enable the Location Services for this app in Settings.')
end

def show_error_message(message)
  alert = UIAlertView.new
  alert.addButtonWithTitle("OK")
  alert.message = message
  alert.show
end
end

```

Figure 22: Code for getting the device location and then displaying the device's current location on a map

5.3.2 Using the camera

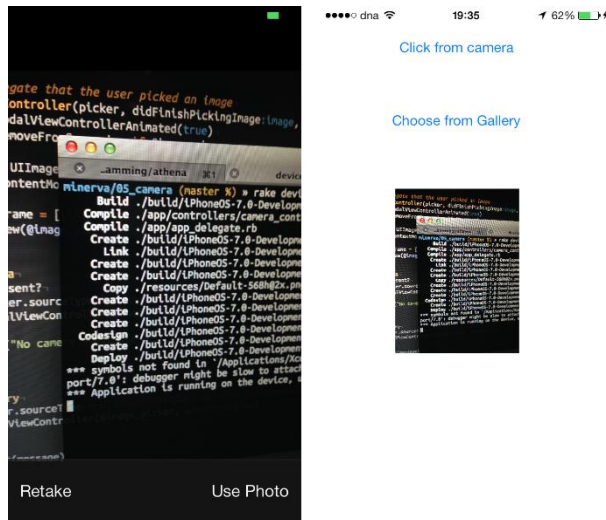


Figure 23: Taking a photo and showing that photo in the application

A modified version of the CameraExample application from the Ruby Essential book was used to initially taking photos using the device camera and picking photos from the gallery.

```
@image_picker = UIImagePickerController.alloc.init
@image_picker.delegate = self
@image_picker.sourceType = UIImagePickerControllerSourceTypeCamera
presentModalViewController(@image_picker, animated:true)
```

Figure 24: Opening the camera interface in a UIViewController only requires four lines of code at its most basic level. This code snippet leaves out checks to make sure that the device in use has a camera present.

5.4 Accessing a remote API

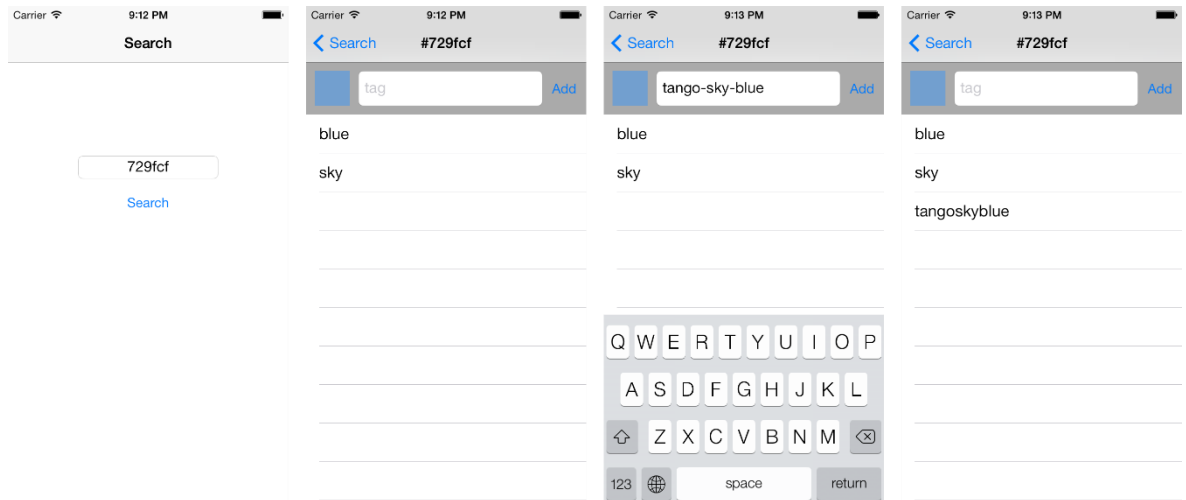


Figure 25: From left-to-right: 1. Entering a search value into the app, 2. The search results, 3. Entering a new tag, 4. After the remote request has completed and the tag has been added

To test creating an app that uses external, HTTP APIs, the “Colr” application from chapter 7 of “RubyMotion: iOS Development with Ruby” was used.

This example application was not fully ready for iOS 7 and the layout especially had to be modified a lot to get it to work as intended on iOS 7.

```
BubbleWrap::HTTP.get("http://www.colr.org/json/color/#{hex}") do |response|
  result_data = BubbleWrap::JSON.parse(response.body.to_str)
  puts color_data.inspect
  block.call(nil)
end
```

Figure 26: Making a GET request to a remote API over HTTP using the BubbleWrap² RubyMotion library. In this example the received data is parsed, printed to the debugging console, then discarded.

² The BubbleWrap library provides Ruby wrappers for iOS SDKs. <https://github.com/rubymotion/BubbleWrap>

The application uses the BubbleWrap Ruby library to interact with a remote HTTP JSON API. Using this library accessing a remote API proved to be extremely simple to do.

5.5 Using 3rd party Objective-C libraries

Objective C libraries can be easily included in RubyMotion applications through the CocoaPods Objective C library dependency manager.

5.5.1 CocoaPods

To test the RubyMotion CocoaPods integration, a new application was created from scratch. The application directly requires two CocoaPod libraries:

“OpenWeatherMapAPP” and “UIImage+PDF”.

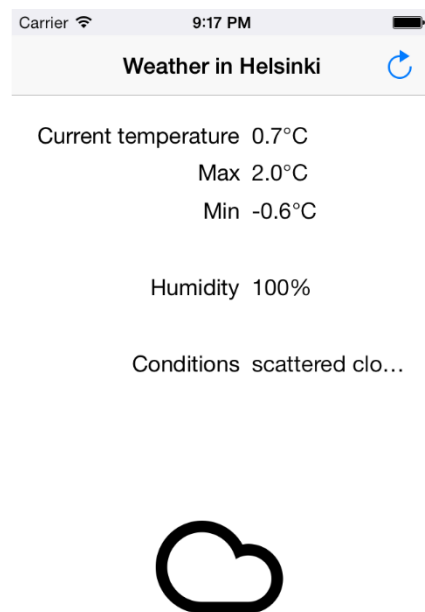


Figure 27: Weather data fetched from OpenWeatherMap and a PDF weather symbol being displayed

The first of these, OpenWeatherMapAPI, is an API wrapper for the OpenWeatherMap API. OpenWeatherMap is a project that provides a free to use weather data API for developers in a similar way in which OpenStreetMap provides free mapping data or the way Wikipedia provides a free encyclopedia. (OpenWeatherMap, 2013)

The UIImage+PDF library allows developers to use PDF images in a similar way as they would normally use raster or bitmap images in their applications. This allows developers to easily use resolution independent, scalable, vector assets in their iOS projects. (Barber, 2011)

```
Motion::Project::App.setup do |app|
  # Use `rake config` to see complete project settings.
  app.name = '07_cocoa_pods'

  app.pods do
    pod "OpenWeatherMapAPI", "~> 0.0.5"
    pod "UIImage+PDF"
  end
end
```

Figure 28: Adding CocoaPod dependencies to a RubyMotion project is extremely easy. Only a couple of lines of code need to be added to the Rakefile

5.6 Interactive testing

RubyMotion ships with an interactive debugger called the REPL (read-eval-print loop). The REPL provides a way to interact with a RubyMotion application that is currently running. For example, the developer can change values in the user interface or use the REPL for debugging an application. (Samsonetti, RubyMotion Project Management Guide, 2013b)

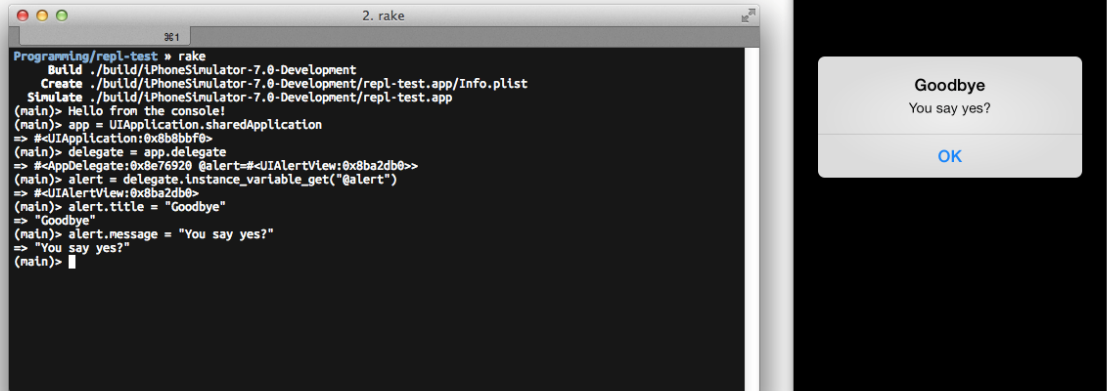


Figure 29: Interactively making changes to an application in the iOS Simulator while it's still running.

An example of changing the user interface while the application is running can be seen in Figure 29. In this case the alert box's title and text were changed while the application was running in the iOS simulator.

The developer can also *command* (⌘ on a Mac keyboard) click views in the simulator to change the context of the REPL console. The developer can then use the `self` object to interact with the selected view in the REPL console. (Samsonetti, RubyMotion Project Management Guide, 2013b)

6 Results, evaluation, and conclusions

6.1 Results

Overall the development of applications using the RubyMotion toolchain was surprisingly pleasant. The major issues encountered are more due to a lack of clear or reliable documentation. Most of the documentation available also hasn't yet been updated to reflect the changes in iOS 7, which was released to the general public on September 18, 2013 (iOS 7 arrives on Apple devices September 18th <http://www.engadget.com/2013/09/10/ios-7-arrives-september-18th/>), about 2 and a half months before the proof of concept development began.

The development process was relatively smooth, despite some small problems. All the areas or features of smartphone application development that were targeted were covered by a test application.

6.2 Evaluation

The goal of this research was to test how RubyMotion could be used by a developer with prior experience with the Ruby programming language, but with no experience developing applications with Objective-C or developing applications for the iOS platform could use RubyMotion as a way of getting into iOS application development.

This objective was met and RubyMotion looks like a viable platform for Ruby developers. There can be a fairly steep learning curve, however, as the iOS SDK can seem foreign to developers with no prior experience in iOS development. As RubyMotion merely wraps the iOS APIs, the Objective-C foundation of iOS application development is never far out of sight.

A successful test or proof of concept application was developed for each feature that was selected to be tested before hand, though problems were encountered in a couple of areas.

6.2.1 Building the application user interface

Two out of three test applications for this section were successful. Both defining views fully programatically and using views built with the graphical Xcode Interface Builder were fully successful.

Though a test case was developed using Auto Layout programatically, this application never functioned fully as intended. This can largely be pinned down to a lack of documentation on how user interfaces are supposed to be built with Auto Layout (when not using Interface Builder).

The Auto Layout features of iOS were successfully used in both the Interface Builder test case and the CocoaPods test case later on.

6.2.2 Storing data locally

This test case was fully successful and no particular problems were encountered. The chosen test application worked fully as intended. While iOS devices have a few locations where data can be stored, this test only tested the most import of these, Core Data.

6.2.3 On-board sensors

Both of the test cases under this feature were successful. Accessing the device location via the iOS SDKs worked as intended. In the other test case, accessing the device camera and photo library was also fully successful.

6.2.4 Accessing a remote API

In the end this test case performed successfully, though there were some issues as the instructions for the chosen test application had not yet been updated to reflect the changes in iOS 7 at the time of the test. In the end the application performed successfully after some modifications.

6.2.5 Using 3rd party Objective-C libraries (via CocoaPods)

In this test application the goal was to see how and how easily a developer using RubyMotion to build iOS applications could use third party Objective-C libraries with their Ruby code.

This test was a success and it proved that using 3rd party libraries through CocoaPods (a dependency manager for Objective-C libraries) is very easy via RubyMotion's built-in CocoaPods integration.

6.3 Conclusions

As a developer who isn't already familiar with coding for iOS, there are also challenges in having to simultaneously learn the internal workings of iOS and the way that RubyMotion itself works. If one was already familiar with iOS development and familiar with the Ruby programming language, picking up RubyMotion for iOS application development would most likely be a lot easier than it is for a developer who is only familiar with one of the two, or none.

As it stands today, based on experience from the proof of concept development, RubyMotion cannot be recommended to anyone who is just getting started with programming. Picking up either Objective C or Ruby development for some other environment (for example web development with the Ruby on Rails framework), would present a more pleasant and more consistent experience for a novice programmer. This is largely due to the fact that developing applications RubyMotion

necessarily mixes concepts and terms from both the iOS “world” and from the Ruby world, for a novice programmer this could be very confusing.

For developers who are already familiar with at least iOS development *or* Ruby development, the experience is less confusing as they will have a more intuitive sense about what concepts and tools belong to which part of the underlying system.

As the use of smartphones and other mobile devices increases worldwide and as we spend more and more time on mobile devices and as we do more things on our mobile devices, RubyMotion can definitely be seen as a valuable tool especially for Ruby programmers looking to get into creative native applications for the iOS ecosystem. Unlike creating applications with Objective C, using RubyMotion doesn't require a Ruby programmer to learn a whole new language and a whole new set of best practices.

7 Summary

The number of smartphones and mobile devices that we (as consumers) buy is increasing at an ever increasing rate. We are also spending a larger amount of time using these devices and we are using them to do more things. Particularly of interest is the fact that more and more money is being spent using these devices. Users are more willing than ever to both buy apps for these devices and more willing to buy other products using these devices. Particularly iOS device users are willing to spend money on apps purchases on their devices.

The Ruby programming language has been moderately successful of late (making it into the top ten or top fifteen languages in terms of popularity). However, until recently it was not possible to use the Ruby language to create applications for any popular smartphone platform. This has relatively recently changed with the introduction of Ruboto for Android and RubyMotion for iOS development.

RubyMotion is a toolchain that enables the development of iOS and OS X applications using the ruby programming language. It is not a bridge, though, rather it is a wrapper for the native iOS SDKs. This means that RubyMotion applications can use all the same features that are available to regular, Objective-C based applications. Developers can also use 3rd party Objective-C libraries in their RubyMotion applications.

While iOS application development with RubyMotion presents its own difficulties over the development of, say, web applications in Ruby, this does not mean that RubyMotion is not a viable option for developers with prior experience in either Ruby or with iOS application development (or both). It might even be possible to share code between, for example, a Ruboto application on Android and an iOS application built with RubyMotion.

At this point RubyMotion does not seem like a platform that should be recommended to novice developers as the fusion of paradigms and concepts from both the world of Ruby and from the iOS SDK can be both confusing and daunting at times.

7.1 Further research questions

While researching this topic, no benchmarks of RubyMotion application performance were found, so this would definitely be an area where more research would be warranted.

Further research could be used to gauge the empirical performance differences between applications built with RubyMotion and applications built with Objective-C. It would also be useful to do some profiling of RubyMotion applications to see if they use more resources (such as CPU or memory) than similar Objective-C applications would. This would also be useful for gauging whether applications built with RubyMotion are more or less stable than applications built with Objective-C.

Another area that could warrant further research is the possibility of sharing Ruby code between different platforms. For example, it may or may not be practically viable to share code between an iOS RubyMotion application, an OS X RubyMotion Application, a Ruboto application running on Android, and a web application using, say, Ruby on Rails.

The test applications were only tested on the latest version of iOS (version 7) and only on a single device and in the simulator. Further research could be done into how well RubyMotion applications run in a more diverse range of devices and iOS versions.

References

- Allan, A. (2013). *Learning iOS Programming* (3rd ed.). Sebastopol: O'Reilly Media, Inc.
- Allsopp, C. (2012a). *RubyMotion: iOS Development with Ruby*. Dallas: The Pragmatic Programmers, LLC.
- Allsopp, C. (2012b, 07 08). *The RubyMotion Way*. Retrieved 11 28, 2013, from clayallsopp.com: <http://clayallsopp.com/posts/the-ruby-motion-way/>
- Apple Inc. (2013a, 10 22). *Start Developing iOS Apps Today*. Retrieved 11 17, 2013, from iOS Developer Library: https://developer.apple.com/library/ios/referencelibrary/GettingStarted/RoadMapiOS/index.html#//apple_ref/doc/uid/TP40011343
- Apple Inc. (2013b). *Data Management in iOS*. Retrieved 11 25, 2013, from Apple Developer Center: <https://developer.apple.com/technologies/ios/data-management.html>
- Apple Inc. (2013c, 10 23). *The iOS Environment*. Retrieved 11 25, 2013, from iOS Developer Library: <https://developer.apple.com/library/ios/documentation/iphone/conceptual/iphoneosprogrammingguide/TheiOSEnvironment/TheiOSEnvironment.html>
- Armando, M. (2012, 09 23). *MacRuby/ Rubymotion Auto Layout Basics*. Retrieved 11 24, 2013, from Mateus - Welt: <http://seanlimateus.github.io/blog/2012/09/23/macruby-slash-rubymotion-auto-layout-basics/>
- Barber, N. T. (2011, 10 16). *UIImage-PDF / README.md*. Retrieved 11 28, 2013, from GitHub UIImage+PDF project git repository: <https://github.com/mindbrix/UIImage-PDF/blob/bbe64f75439ce20aabc3a664c4ae506c540bf4b3/README.md>
- Bradley, T. (2013, 11 15). *Android Dominates Market Share, But Apple Makes All The Money*. Retrieved 11 21, 2013, from Forbes: <http://www.forbes.com/sites/tonybradley/2013/11/15/android-dominates-market-share-but-apple-makes-all-the-money/>

- Burnette, E. (2010). *Hello, Android* (3rd ed.). Dallas: Pragmatic Programmers, LLC.
- Carsten, B. (1989, 11). Carsten's Corner. *Power Conversion and Intelligent Motion*, p. 38.
- comScore. (2013, 10 21). *1 in 7 European Smartphone Owners Make Online Purchases via their Device*. Retrieved 10 28, 2013, from comScore web site:
http://www.comscore.com/Insights/Press_Releases/2013/10/1_in_7_European_Smartphone_Owners_Make_Online_Purchases_via_their_Device
- Dilger, D. E. (2013, 11 12). *IDC data shows 66% of Android's 81% smartphone share are junk phones selling for \$215*. Retrieved 11 21, 2012, from AppleInsider:
<http://appleinsider.com/articles/13/11/12/idc-data-shows-66-of-androids-81-smartphone-share-are-junk-phones-selling-for-215>
- Dornin, L., Mednieks, Z., Meike, G. B., & Nakamura, M. (2012). *Programming Android* (2nd ed.). Sebastopol: O'Reilly Media, Inc.
- eMarketer. (2013, 11 13). *Mobile Set to Take Double-Digit Share of Western Europe's Online Holiday Sales*. Retrieved 11 16, 2013, from eMarketer web site:
<http://www.emarketer.com/Article/Mobile-Set-Take-Double-Digit-Share-of-Western-Europes-Online-Holiday-Sales/1010375>
- HipByte. (2013, 10 28). *RubyMotion Features*. Retrieved 10 28, 2013, from Official Rubymotion web site: <http://www.rubymotion.com/features/>
- Nalwaya, A., & Paul, A. (2013). *RubyMotion iOS Development Essentials*. Birmingham: Packt Publishing Ltd.
- Nielsen. (2010, 09 14). *The state of mobile apps*. Retrieved 10 28, 2013, from Nielsen web site: <http://www.nielsen.com/us/en/reports/2010/The-State-Of-Mobile-Apps.html>
- Nielsen. (2013, 10 29). *Ring the bells: More smartphones in students' hands ahead of back-to-school season*. Retrieved 11 07, 2013, from Nielsen web site:
<http://www.nielsen.com/us/en/newswire/2013/ring-the-bells-more-smartphones-in-students-hands-ahead-of-back.html>
- Nutter, C. (2009, 02 25). *Ruboto Is Your Friend*. Retrieved 11 17, 2013, from Headius blog: <http://blog.headius.com/2009/02/ruboto-is-your-friend.html>

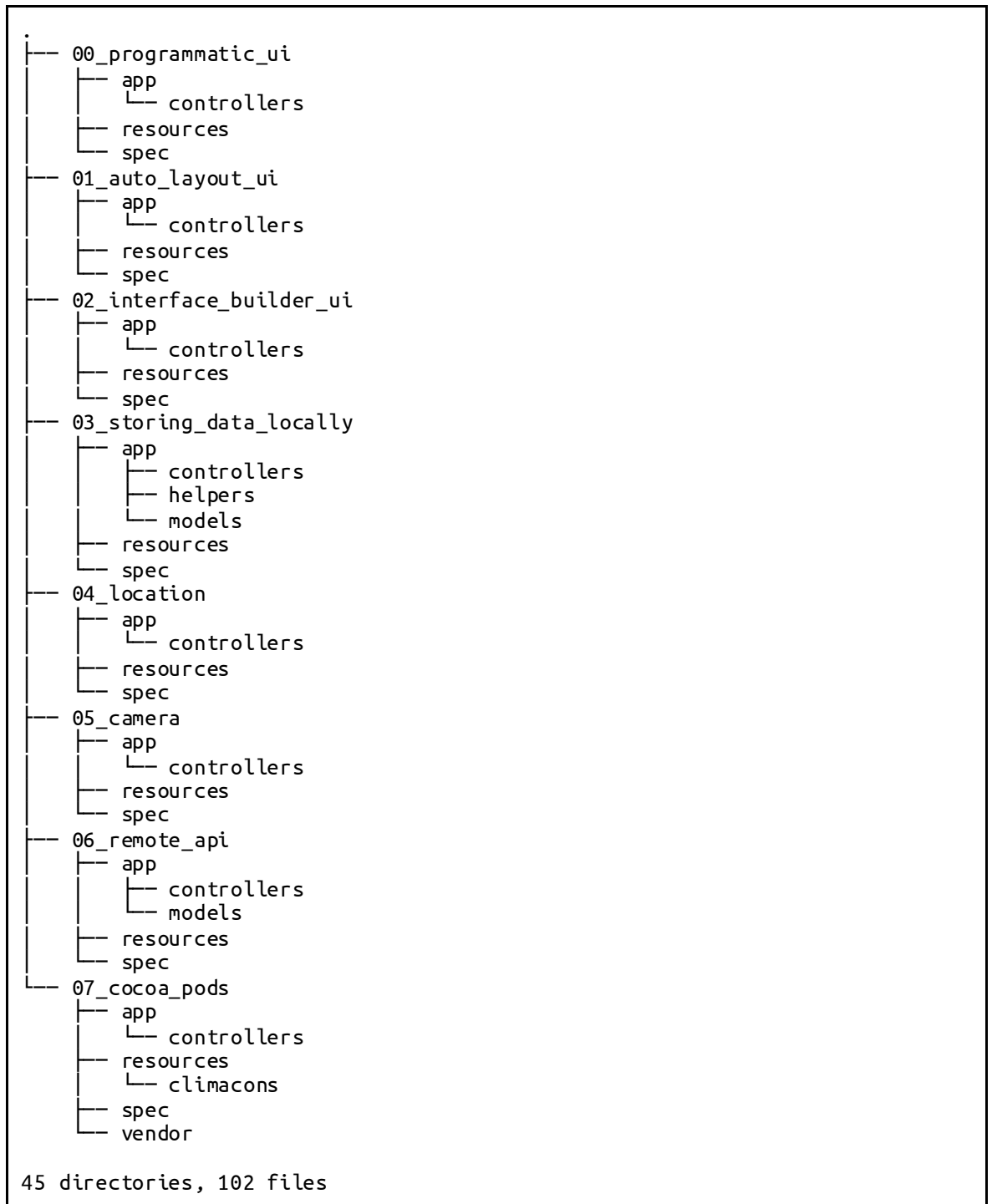
- OPA. (2012, 08 01). *A Portrait of Today's Smartphone User*. Retrieved 11 21, 2013, from Online Publisher's Association:
[http://onlinepubs.ehclients.com/images/pdf/MMF-OPA_-_Portrait_of_Smartphone_User_-_Aug12_\(Public\).pdf](http://onlinepubs.ehclients.com/images/pdf/MMF-OPA_-_Portrait_of_Smartphone_User_-_Aug12_(Public).pdf)
- OpenWeatherMap. (2013). *OpenWeatherMap*. Retrieved 11 28, 2013, from OpenWeatherMap web site: <http://openweathermap.org/>
- Paul, R. (2012, 05 03). *Exclusive: building native iOS apps with RubyMotion*. Retrieved 11 17, 2013, from Ars Technica: Exclusive: building native iOS apps with RubyMotion
- Petzold, C. (2010). *Programming Windows Phone 7*. Redmond: Microsoft Press.
- Phillips, I. (2012, 05 07). *RubyMotion and Interface Builder...* Retrieved 11 24, 2013, from Digital Magpie: <http://ianp.org/2012/05/07/rubymotion-and-interface-builder/>
- RedMonk. (2013, 02 28). *The RedMonk Programming Language Rankings: January 2013*. Retrieved 11 17, 2013, from RedMonk web site:
<http://redmonk.com/sograzy/2013/02/28/language-rankings-1-13/>
- Samsonetti, L. (2008, 04 14). *[ANN] MacRuby 0.1*. Retrieved 10 18, 2013, from Ruby-Talk mailing list: <http://blade.nagaokaut.ac.jp/cgi-bin/scat.rb/ruby/ruby-talk/294485>
- Samsonetti, L. (2013a, 05 10). *Welcome to RubyMotion*. Retrieved 11 23, 2013, from RubyMotion Developer Center: <http://www.rubymotion.com/developer-center/guides/getting-started/>
- Samsonetti, L. (2013b, 06 10). *RubyMotion Project Management Guide*. Retrieved 11 23, 2013, from RubyMotion Developer Center:
http://www.rubymotion.com/developer-center/guides/project-management/#_install_on_device
- Suri, I. (2013, 05 13). *RubyMotion 2.0 Arrives with OS X Support, Templates and Plug-ins*. Retrieved 11 17, 2013, from DevOps Anfile:
<http://devopsangle.com/2013/05/13/rubymotion-2-0-arrives-with-os-x-support-templates-and-plug-ins/>

- Symonds, J. (2012, 05 04). *Why RubyMotion Is Better Than Objective-C*. Retrieved 11 21, 2011, from Hi, I'm Josh Symonds:
<http://joshsymonds.com/blog/2012/05/04/why-rubymotion-is-better-than-objective-c/>
- The MacRuby Team. (2013, 01 25). *MacRuby*. Retrieved 10 28, 2013, from MacRuby web site: <http://macruby.org/>
- The Transparent Language Popularity Index. (2013, 07 01). *Results: July 2013 update*. Retrieved 11 17, 2013, from The Transparent Language Popularity Index web site: <http://lang-index.sourceforge.net/#grid>
- TIOBE Software. (2013, 11 10). *TIOBE Programming Community Index for November 2013*. Retrieved 11 17, 2013, from TIOBE web site:
<http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>
- Zaph. (2013, 10 25). *Xcode 4.4 and later install Command Line Tools*. Retrieved 11 23, 2013, from Stack Overflow: <http://stackoverflow.com/a/9329325/1439994>

Attachments

Attachment 1: The directory tree of the proof of concept applications

Note: Only directories are listed, not files.



Attachment 2: 00_programmatic_ui/Rakefile

```
# -*- coding: utf-8 -*-
$.unshift("/Library/RubyMotion/lib")
require 'motion/project/template/ios'

begin
  require 'bundler'
  Bundler.require
rescue LoadError
end

Motion::Project::App.setup do |app|
  # Use `rake config` to see complete project settings.
  app.name = '00_programmatic_ui'
end
```

Attachment 3: 00_programmatic_ui/app/app_delegate.rb

```
class AppDelegate
  def application(application, didFinishLaunchingWithOptions:launchOptions)
    @window = UIWindow.alloc.initWithFrame(UIScreen.mainScreen.bounds)
    @window.makeKeyAndVisible

    controller = ColorsController.alloc.initWithNibName(nil, bundle: nil)
    nav_controller =
  UINavigationController.alloc.initWithRootViewController(controller)
    tab_controller = UITabBarController.alloc.initWithNibName(nil, bundle: nil)

    top_controller = ColorDetailController.alloc.initWithColor(UIColor.purpleColor)
    top_controller.title = "Top Color"
    top_nav_controller =
  UINavigationController.alloc.initWithRootViewController(top_controller)

    tab_controller.viewControllers = [nav_controller, top_nav_controller]

    @window.rootViewController = tab_controller

    true
  end
end
```

Attachment 4: 00_programmatic_ui/app/controllers/change_color_controller.rb

```
class ChangeColorController < UIViewController
  attr_accessor :color_detail_controller

  def viewDidLoad
    super

    self.title = "Change Color"
    self.view.backgroundColor = UIColor.whiteColor

    @text_field = UITextField.alloc.initWithFrame(CGRectZero)
    @text_field.borderStyle = UITextBorderStyleRoundedRect
    @text_field.textAlignment = UITextAlignmentCenter
    @text_field.placeholder = "Enter a color"

    @text_field.frame = [CGPointZero, [150, 32]]
    @text_field.center = [self.view.frame.size.width / 2, self.view.frame.size.height
/ 2 - 170]
    self.view.addSubview(@text_field)

    @button = UIButton.buttonWithType(UIButtonTypeRoundedRect)
    @button.setTitle("Change", forState: UIControlStateNormal)
    @button.frame = [
      [
        @text_field.frame.origin.x,
        @text_field.frame.origin.y + @text_field.frame.size.height + 10
      ],
      @text_field.frame.size
    ]
    self.view.addSubview(@button)
    @button.addTarget(self,
      action:"change_color",
      forControlEvents:UIControlEventTouchUpInside)
  end

  def change_color(*args)
    color_text = @text_field.text.downcase
    color_method = "#{color_text}Color"

    if UIColor.respond_to?(color_method)
      color = UIColor.send("#{color_text}Color")
    else
      @text_field.text = "Error!"
      return
    end

    self.color_detail_controller.view.backgroundColor = color
    self.dismissViewControllerAnimated(true, completion:lambda {})
  end
end
```

Attachment 4: 00_programmatic_ui/app/controllers/color_detail_controller.rb

```
class ColorDetailController < UIViewController
  def initWithColor(color)
    self.initWithNibName(nil, bundle:nil)
    self.tabBarItem = UITabBarItem.alloc.initWithTitle("Color detail", image:
UIImage.imageNamed("12-eye.png"), tag: 1)
    @color = color
    self
  end

  def viewDidLoad
    super
    self.view.backgroundColor = @color
    self.title = "Detail"

    rightButton = UIBarButtonItem.alloc.initWithTitle("Change",
      style: UIBarButtonItemStyleBordered,
      target: self,
      action: "change_color")
    self.navigationItem.rightBarButtonItem = rightButton
  end

  def change_color
    controller = ChangeColorController.alloc.initWithNibName(nil, bundle:nil)
    controller.color_detail_controller = self
    self.presentViewController(
      UINavigationController.alloc.initWithRootViewController(controller),
      animated:true,
      completion: lambda {}
    )
  end
end
```

Attachment 5: 01_auto_layout/app/Rakefile

```
# -*- coding: utf-8 -*-
$.unshift("/Library/RubyMotion/lib")
require 'motion/project/template/ios'

begin
  require 'bundler'
  Bundler.require
rescue LoadError
end

Motion::Project::App.setup do |app|
  # Use `rake config` to see complete project settings.
  app.name = '01_auto_layout_ui'
end
```

Attachment 6: 01_auto_layout/app/app_delegate.rb

```
class AppDelegate
  def application(application, didFinishLaunchingWithOptions:launchOptions)
    @window = UIWindow.alloc.initWithFrame(UIScreen.mainScreen.bounds)
    @window.makeKeyAndVisible

    controller = ColorsController.alloc.initWithNibName(nil, bundle: nil)
    nav_controller =
  UINavigationController.alloc.initWithRootViewController(controller)
    tab_controller = UITabBarController.alloc.initWithNibName(nil, bundle: nil)

    top_controller = ColorDetailController.alloc.initWithColor(UIColor.purpleColor)
    top_controller.title = "Top Color"
    top_nav_controller =
  UINavigationController.alloc.initWithRootViewController(top_controller)

    tab_controller.viewControllers = [nav_controller, top_nav_controller]

    @window.rootViewController = tab_controller

    true
  end
end
```


Attachment 7: 01_auto_layout/app/controllers/colors_controller.rb

```
class ColorsController < UIViewController
  def viewDidLoad
    super

    self.view.backgroundColor = UIColor.whiteColor
    self.title = "Colors"

    @label = UILabel.new
    @label.text = "Colors"
    @label.textAlignment = UITextAlignmentCenter

    subviews = { "label" => @label }

    %w(red green blue).map do |color_text, index|
      color = UIColor.send("#{color_text}Color")
      button = UIButton.buttonWithType(UIButtonTypeRoundedRect)
      button.setTitle(color_text, forState: UIControlStateNormal)
      button.setTitleColor(color, forState: UIControlStateNormal)
      button.sizeToFit
      button.addTarget(self,
        action: "tap_#{color_text}",
        forControlEvents: UIControlEventTouchUpInside)

      subviews["#{color_text}_button"] = button
    end

    Motion::Layout.new do |layout|
      layout.view self.view
      layout.subviews subviews
      layout.metrics "top" => 150, "margin" => 20, "height" => 40
      layout.vertical "|-top-[label(==height)]-margin-[red_button(==height)]-margin-[green_button(==height)]-margin-[blue_button]"
      layout.horizontal "|-margin-[label]-margin-|"
      layout.horizontal "|-margin-[red_button]-margin-|"
      layout.horizontal "|-margin-[green_button]-margin-|"
      layout.horizontal "|-margin-[blue_button]-margin-|"
    end
  end

  def initWithNibName(name, bundle: bundle) super
    self.tabBarItem = UITabBarItem.alloc.initWithTitle("Colors", image:
      UIImage.imageNamed("173-eyedropper.png"), tag: 1)
    self
  end

  def tap_red
    controller = ColorDetailController.alloc.initWithColor(UIColor.redColor)
    self.navigationController.pushViewController(controller, animated: true)
  end

  def tap_green
    controller = ColorDetailController.alloc.initWithColor(UIColor.greenColor)
    self.navigationController.pushViewController(controller, animated: true)
  end

  def tap_blue
    controller = ColorDetailController.alloc.initWithColor(UIColor.blueColor)
    self.navigationController.pushViewController(controller, animated: true)
  end
end
```

Attachment 8: 02_interface_builder_ui/app/controllers/colors_controller.rb

```
class ColorsController < UIViewController
  def loadView
    views = NSBundle.mainBundle.loadNibNamed "colors", owner: self, options: nil
    self.view = views[0]
    @view_handle = self.view
  end

  def viewDidLoad
    super

    self.title = "Colors"

    @label = view.viewWithTag 1
    red_button = view.viewWithTag 2
    green_button = view.viewWithTag 3
    blue_button = view.viewWithTag 4

    red_button.addTarget(self, action: 'tap_red',
forControlEvents: UIControlEventTouchUpInside)
    green_button.addTarget(self, action: 'tap_green',
forControlEvents: UIControlEventTouchUpInside)
    blue_button.addTarget(self, action: 'tap_blue',
forControlEvents: UIControlEventTouchUpInside)
  end

  def initWithNibName(name, bundle: bundle) super
    self.tabBarItem = UITabBarItem.alloc.initWithTitle("Colors", image:
UIImage.imageNamed("173-eyedropper.png"), tag: 1)
    self
  end

  def tap_red
    controller = ColorDetailController.alloc.initWithColor(UIColor.redColor)
    self.navigationController.pushViewController(controller, animated: true)
  end

  def tap_green
    controller = ColorDetailController.alloc.initWithColor(UIColor.greenColor)
    self.navigationController.pushViewController(controller, animated: true)
  end

  def tap_blue
    controller = ColorDetailController.alloc.initWithColor(UIColor.blueColor)
    self.navigationController.pushViewController(controller, animated: true)
  end
end
```

Attachment 9: 02_interface_builder_ui/resources/colors.xib

```

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<document type="com.apple.InterfaceBuilder3.CocoaTouch.XIB" version="3.0" toolsVersion="4514"
systemVersion="13A603" targetRuntime="iOS.CocoaTouch" propertyAccessControl="none" useAutolayout="YES">
  <dependencies>
    <plugin identifier="com.apple.InterfaceBuilder.IBCocoaTouchPlugin" version="3747"/>
  </dependencies>
  <objects>
    <placeholder placeholderIdentifier="IBFilesOwner" id="-1" userLabel="File's Owner"/>
    <placeholder placeholderIdentifier="IBFirstResponder" id="-2" customClass="UIResponder"/>
    <view contentMode="scaleToFill" id="1">
      <rect key="frame" x="0.0" y="0.0" width="320" height="568"/>
      <autoresizingMask key="autoresizingMask" widthSizable="YES" heightSizable="YES"/>
      <subviews>
        <label opaque="NO" clipsSubviews="YES" userInteractionEnabled="NO" tag="1"
contentMode="left" horizontalHuggingPriority="251" verticalHuggingPriority="251" ambiguous="YES"
misplaced="YES" text="Colors" lineBreakMode="tailTruncation" baselineAdjustment="alignBaselines"
adjustsFontSizeToFit="NO" translatesAutoresizingMaskIntoConstraints="NO" id="k4h-Td-2xa">
          <rect key="frame" x="135" y="273" width="51" height="21"/>
          <autoresizingMask key="autoresizingMask" flexibleMaxX="YES" flexibleMaxY="YES"/>
          <fontDescription key="fontDescription" type="system" pointSize="17"/>
          <color key="textColor" cocoaTouchSystemColor="darkTextColor"/>
          <nil key="highlightedColor"/>
        </label>
        <button opaque="NO" tag="2" contentMode="scaleToFill" ambiguous="YES" misplaced="YES"
contentHorizontalAlignment="center" contentVerticalAlignment="center" buttonType="roundedRect"
lineBreakMode="middleTruncation" translatesAutoresizingMaskIntoConstraints="NO" id="Saz-aK-zmI">
          <rect key="frame" x="3" y="358" width="80" height="30"/>
          <autoresizingMask key="autoresizingMask" flexibleMaxX="YES" flexibleMaxY="YES"/>
          <state key="normal" title="Red">
            <color key="titleColor" red="1" green="0.0" blue="0.0" alpha="1"
colorSpace="calibratedRGB"/>
            <color key="titleShadowColor" white="0.5" alpha="1" colorSpace="calibratedWhite"/>
          </state>
        </button>
        <button opaque="NO" tag="3" contentMode="scaleToFill" ambiguous="YES" misplaced="YES"
contentHorizontalAlignment="center" contentVerticalAlignment="center" buttonType="roundedRect"
lineBreakMode="middleTruncation" translatesAutoresizingMaskIntoConstraints="NO" id="dzW-5f-JgW">
          <rect key="frame" x="120" y="358" width="80" height="30"/>
          <autoresizingMask key="autoresizingMask" flexibleMaxX="YES" flexibleMaxY="YES"/>
          <state key="normal" title="Green">
            <color key="titleColor" red="0.0" green="1" blue="0.0" alpha="1"
colorSpace="calibratedRGB"/>
            <color key="titleShadowColor" white="0.5" alpha="1" colorSpace="calibratedWhite"/>
          </state>
        </button>
        <button opaque="NO" tag="4" contentMode="scaleToFill" ambiguous="YES" misplaced="YES"
contentHorizontalAlignment="center" contentVerticalAlignment="center" buttonType="roundedRect"
lineBreakMode="middleTruncation" translatesAutoresizingMaskIntoConstraints="NO" id="mge-Yd-hxJ">
          <rect key="frame" x="237" y="358" width="80" height="30"/>
          <autoresizingMask key="autoresizingMask" flexibleMaxX="YES" flexibleMaxY="YES"/>
          <state key="normal" title="Blue">
            <color key="titleColor" red="0.0" green="0.0" blue="1" alpha="1"
colorSpace="calibratedRGB"/>
            <color key="titleShadowColor" white="0.5" alpha="1" colorSpace="calibratedWhite"/>
          </state>
        </button>
      </subviews>
      <color key="backgroundColor" white="1" alpha="1" colorSpace="custom"
customColorSpace="calibratedWhite"/>
      <constraints>
        <constraint firstItem="k4h-Td-2xa" firstAttribute="top" secondItem="1"
secondAttribute="top" constant="80" id="32U-Oi-FUo"/>
        <constraint firstItem="Saz-aK-zmI" firstAttribute="width" secondItem="mge-Yd-hxJ"
secondAttribute="width" id="Mqc-kq-gML"/>
        <constraint firstItem="Saz-aK-zmI" firstAttribute="top" secondItem="dzW-5f-JgW"
secondAttribute="top" id="Wvs-28-dbv"/>
        <constraint firstItem="Saz-aK-zmI" firstAttribute="top" secondItem="mge-Yd-hxJ"
secondAttribute="top" id="c30-kP-rUa"/>
        <constraint firstItem="dzW-5f-JgW" firstAttribute="top" secondItem="k4h-Td-2xa"
secondAttribute="bottom" constant="80" id="d5L-DG-zcz"/>
        <constraint firstItem="Saz-aK-zmI" firstAttribute="width" secondItem="dzW-5f-JgW"
secondAttribute="width" id="qMV-oZ-sws"/>
      </constraints>
      <simulatedStatusBarMetrics key="simulatedStatusBarMetrics"/>
      <simulatedScreenMetrics key="simulatedDestinationMetrics" type="retina4"/>
    </view>
  </objects>
</document>

```

Attachment 10: 03_storing_data_locally/Rakefile

```
# -*- coding: utf-8 -*-
$.unshift("/Library/RubyMotion/lib")
require 'motion/project/template/ios'

begin
  require 'bundler'
  Bundler.require
rescue LoadError
end

Motion::Project::App.setup do |app|
  # Use `rake config` to see complete project settings.
  app.name = '03_storing_data_locally'
  app.frameworks += %w( CoreData )
end
```

Attachment 11: 03_storing_data_locally/app/app_delegate.rb

```
class AppDelegate
  def application(application, didFinishLaunchingWithOptions:launchOptions)
    setting_core_data

    employee_view_controller = EmployeeViewController.alloc.init

    # We need to pass the Managed Object Context to the next controller so we can use
    it later for creating, fetching or deleting objects
    employee_view_controller.managed_object_context = @managed_object_context

    @window = UIWindow.alloc.initWithFrame(UIScreen.mainScreen.bounds)
    @window.rootViewController =
    UINavigationController.alloc.initWithRootViewController(employee_view_controller)
    @window.makeKeyAndVisible

    true
  end

  def setting_core_data
    # First we need to create the NSManagedObjectContext with all the entities and
    their relationships.
    managed_object_model = NSManagedObjectContext.alloc.init
    managed_object_model.entities = [Employee.entity]

    # The next object needed is the NSPersistentStoreCoordinator which will allow
    Core Data to persist the information.
    persistent_store_coordinator =
    NSPersistentStoreCoordinator.alloc.initWithManagedObjectContext(managed_object_model)

    # Now lets get a URL for where we want Core Data to create the persist file, in
    this case a SQLite Database File
    persistent_store_file_url = NSURL.fileURLWithPath(File.join(NSHomeDirectory(),
    "Documents", "EmployeeStore.sqlite"))

    error_pointer = Pointer.new(:object)

    # Add a new Persistent Store to our Persistent Store Coordinator which means that
    we are telling the Persistent Store Coordinator where to perform the save of our
    objects.
    # In this case we are stating that our objects must be stored in a SQLite
    database in the path we already created previously
    unless persistent_store_coordinator.addPersistentStoreWithType(NSSQLiteStoreType,
    configuration:nil, URL:persistent_store_file_url, options:nil, error:
    error_pointer)
      # In case we can't initialize the Persistence Store File
      raise "Cannot initialize Core Data Persistence Store Coordinator:
      #{error_pointer[0].description}"
    end

    # Finally our most important object, the Managed Object Context, is responsible
    for creating, destroying, and fetching the objects
    @managed_object_context = NSManagedObjectContext.alloc.init
    @managed_object_context.persistentStoreCoordinator = persistent_store_coordinator
  end
end
```

Attachment 12: 03_storing_data_locally/app/controllers/add_employee_view_controller.rb

```
class AddEmployeeViewController < UIViewController
  attr_accessor :managed_object_context
  def viewDidLoad
    self.view.backgroundColor = UIColor.whiteColor
    self.title = 'Add Employee'
    save_bar_button_item = UIBarButtonItem.alloc.initWithTitle('Save', style:
UIBarButtonItemStyleDone, target: self, action: 'save_employee')
    self.navigationItem.rightBarButtonItem = save_bar_button_item
    load_form
  end

  def save_employee
    # Using Core Data create a new instance of the object employee
    employee = NSEntityDescription.insertNewObjectForEntityForName(Employee.name,
inManagedObjectContext: @managed_object_context)

    # Assign the text of the name text field to the employee
    employee.name = @name.text
    employee.age = @age.text.intValue

    # Create a new pointer for managing the errors
    error_pointer = Pointer.new(:object)

    # Lets persist the new Movie object, saving the managed object context that
contains it
    unless @managed_object_context.save(error_pointer)
      raise "Error saving a new Director: #{error_pointer[0].description}"
    end

    # Pop the Director View Controller
    self.navigationController.popViewControllerAnimated(true)
  end

  def load_form
    @name = UITextField.alloc.initWithFrame([[50,100], [200,30]])
    @name.borderStyle = UITextBorderStyleRoundedRect
    @name.placeholder = "Name"
    self.view.addSubview(@name)

    @age = UITextField.alloc.initWithFrame([[50,150], [200,30]])
    @age.borderStyle = UITextBorderStyleRoundedRect
    @age.keyboardType = UIKeyboardTypeNumberPad
    @age.placeholder = "Age"
    self.view.addSubview(@age)
  end
end
```

Attachment 13: 03_storing_data_locally/ app/controllers/employee_view_controller.rb

```
class EmployeeViewController < UIViewController
  attr_accessor :managed_object_context

  def loadView
    # Set up the title for the View Controller
    self.title = "Employee"
    # Create a new TableView for showing the Text Fields
    table_view = UITableView.alloc.initWithFrame(UIScreen.mainScreen.bounds, style:
UITableViewStyleGrouped)
    table_view.dataSource = self
    self.view = table_view

    # Create a new Bar Button Item with the Add System Default
    add_new_employee_item =
UIBarButtonItem.alloc.initWithBarButtonSystemItem(UIBarButtonSystemItemAdd, target:
self, action: "add_new_employee")

    # Add the Bar Button Item to the Navigation Bar
    self.navigationItem.rightBarButtonItem = add_new_employee_item
  end

  def viewWillAppear(animated)
    super
    reload_data
  end

  def reload_data
    fetch_request = NSFetchRequest.alloc.init
    entity = NSEntityDescription.entityForName(Employee.name, inManagedObjectContext:
@managed_object_context)
    fetch_request.setEntity(entity)

    # Sort the Employee by employee name
    fetch_sort = NSSortDescriptor.alloc.initWithKey('name', ascending: true)
    fetch_request.setSortDescriptors([fetch_sort])

    # Update the fetch employee array and reload the table view
    update_fetched_employee_with_fetch_request(fetch_request)
  end

  def update_fetched_employee_with_fetch_request(fetch_request)
    # Create a new pointer for managing the errors
    error_pointer = Pointer.new(:object)
    # Using the NSManagedObjectContext execute the fetch request
    @fetched_employee = @managed_object_context.executeFetchRequest(fetch_request,
error: error_pointer)

    # If the returning array of the fetch request is nil
    # means that a problem has occurred
    unless @fetched_employee
      raise "Error fetching employee: #{error_pointer[0].description}"
    end

    # refresh table view to reload its data
    self.view.reloadData
  end

  # UITableView Data Source
  def tableView(tableView, numberOfRowsInSection: section)
    @fetched_employee.count
  end
end
```

```

def tableView(tableView, cellForRowAtIndexPath: indexPath)
  cell_identifier = 'EmployeeCell'
  cell = tableView.dequeueReusableCellWithIdentifier(cell_identifier)

  # If we are not cells to use we need to create one
  if cell == nil
    # Lets create a new UITableViewCell with the identifier
    cell = UITableViewCell.alloc.initWithStyle(UITableViewCellStyleValue1,
reuseIdentifier:cell_identifier)
    cell.selectionStyle = UITableViewCellSelectionStyleNone
  end

  employee = @fetched_employee[indexPath.row]
  cell.textLabel.text = employee.name
  cell.detailTextLabel.text = employee.age.to_s
  cell
end

def tableView(tableView, canEditRowAtIndexPath: indexPath)
  true
end

def tableView(tableView, commitEditingStyle: editingStyle, forRowAtIndexPath:
indexPath)
  employee = @fetched_employee[indexPath.row]

  # Ask the NSManagedObjectContext to delete the object
  @managed_object_context.deleteObject(employee)

  # Create a new pointer for managing the errors
  error_pointer = Pointer.new(:object)

  # Lets persist the deleted employee object, saving the managed object context
that contains it
  unless @managed_object_context.save(error_pointer)
    raise "Error deleting an Employee: #{error_pointer[0].description}"
  end

  # Create a new mutable copy of the fetched_employee array
  mutable_fetched_employee = @fetched_employee.mutableCopy

  # Remove the employee from the array
  mutable_fetched_employee.delete(employee)

  # Assign the modified array to our fetched_employee property
  @fetched_employee = mutable_fetched_employee

  # Tell the table view to delete the row
  tableView.deleteRowsAtIndexPaths([indexPath],
withRowAnimation:UITableViewRowAnimationFade)
end

def add_new_employee
  add_employee_view_controller = AddEmployeeViewController.alloc.init

  # We need to pass the Managed Object Context to the next controller so we can use
it later for creating, fetching or deleting objects
  add_employee_view_controller.managed_object_context = @managed_object_context
  self.navigationController.pushViewController(add_employee_view_controller,
animated:true)
end
end

```

Attachment 14: 03_storing_data_locally/app/helpers/ns_entity_description.rb


```

class NSEntityDescription
  def self.newEntityDescriptionWithName(name, attributes:attributes)
    entity = self.alloc.init
    entity.name = name
    entity.managedObjectClassName = name
    attributes = attributes.each.map do |name, type, default, optional, transient,
indexed|
      property = NSAttributeDescription.alloc.init
      property.name = name
      property.attributeType = type
      property.defaultValue = default if default != nil
      property.optional = optional
      property.transient = transient
      property.indexed = indexed
      property
    end
    entity.properties = attributes
  end
end

```

Attachment 15: 03_storing_data_locally/app/helpers/ns_managed_object.rb

```

class NSManagedObject
  def self.entity
    @entity ||= NSEntityDescription.newEntityDescriptionWithName(name, attributes:
@attributes)
  end

  def self.objects
    # Use if you do not want any section in your table view
    @objects ||= NSFetchRequest.fetchObjectsForEntity
      ForName(name, withSortKey: @sortKey, ascending: false, inManagedObjectContext:
Store.shared.context)
  end
end

```

Attachment 16: 03_storing_data_locally/app/models/employee.rb

```

class Employee < NSManagedObject
  # Attribute Name, Data Type, Default Value, Is Optional, Is Transient, Is Indexed
  @attributes ||= [
    ["name", NSStringAttributeType, "", false, false, false],
    ["age", NSInteger32AttributeType, 0, false, false, false]
  ]
end

```

Attachment 17: 04_location/Rakefile

```
# -*- coding: utf-8 -*-
$.unshift("/Library/RubyMotion/lib")
require 'motion/project/template/ios'

begin
  require 'bundler'
  Bundler.require
rescue LoadError
end

Motion::Project::App.setup do |app|
  # Use `rake config` to see complete project settings.
  app.name = '04_location'
  app.frameworks = ['CoreLocation', 'MapKit']
end
```

Attachment 18: 04_location/app/app_delegate.rb

```
class AppDelegate
  def application(application, didFinishLaunchingWithOptions:launchOptions)
    @window = UIWindow.alloc.initWithFrame(UIScreen.mainScreen.bounds)
    @window.rootViewController = LocationController.alloc.init
    @window.makeKeyAndVisible
    true
  end
end
```

Attachment 19: 04_location/app/controllers/location_controller.rb

```
class LocationController < UIViewController
  def viewDidLoad
    view.backgroundColor = UIColor.whiteColor
    self.title = "Location"

    check_location
    show_map
  end

  def check_location
    if (CLLocationManager.locationServicesEnabled)
      @location_manager = CLLocationManager.alloc.init
      @location_manager.desiredAccuracy = KCLLocationAccuracyKilometer
      @location_manager.delegate = self
      @location_manager.purpose = "This application's functionality is based on your
current location "
      @location_manager.startUpdatingLocation
    else
      show_error_message('Please enable the Location Services for this app in
Settings.')
```

```
    end
```

```
  end
```

```
  def show_map
```

```
    map = MKMapView.alloc.initWithFrame( [[0, 20], [320, 568]] )
```

```
    map.mapType = MKMapTypeStandard
```

```
    map.showsUserLocation = true
```

```
    location = CLLocationCoordinate2D.new(@latitude.to_f, @longitude.to_f)
```

```
    map.setRegion(MKCoordinateRegionMake(location, MKCoordinateSpanMake(1, 1)),
animated:true)
```

```
    self.view.addSubview(map)
```

```
  end
```

```
  def locationManager(manager, didUpdateToLocation: newLocation, fromLocation:
oldLocation)
```

```
    @latitude = newLocation.coordinate.latitude
```

```
    @longitude = newLocation.coordinate.longitude
```

```
    @location_manager.stopUpdatingLocation
```

```
    show_map
```

```
  end
```

```
  def locationManager(manager, didFailWithError:error)
```

```
    show_error_message('Please enable the Location Services for this app in
Settings.')
```

```
  end
```

```
  def show_error_message(message)
```

```
    alert = UIAlertView.new
```

```
    alert.addButtonWithTitle("OK")
```

```
    alert.message = message
```

```
    alert.show
```

```
  end
```

```
end
```

Attachment 20: 05_camera/app/controllers/camera_controller.rb

```
class CameraController < UIViewController

  def viewDidLoad
    view.backgroundColor = UIColor.whiteColor
    load_view
  end

  def load_view
    @camera_button = UIButton.buttonWithType(UIButtonTypeRoundedRect)
    @camera_button.frame = [[50, 20], [200, 50]]
    @camera_button.setTitle("Click from camera", forState:UIControlStateNormal)
    @camera_button.addTarget(self, action: :start_camera,
forControlEvents:UIControlEventTouchUpInside)
    view.addSubview(@camera_button)

    @gallery_button = UIButton.buttonWithType(UIButtonTypeRoundedRect)
    @gallery_button.frame = [[50, 100], [200, 50]]
    @gallery_button.setTitle("Choose from Gallery", forState:UIControlStateNormal)
    @gallery_button.addTarget(self, action: :open_gallery,
forControlEvents:UIControlEventTouchUpInside)
    view.addSubview(@gallery_button)

    @image_picker = UIImagePickerController.alloc.init
    @image_picker.delegate = self
  end

  # Tells the delegate that the user picked an image
  def imagePickerController(picker, didFinishPickingImage:image, editingInfo:info)
    self.dismissModalViewControllerAnimated(true)
    @image_view.removeFromSuperview if @image_view

    @image_view = UIImageView.alloc.initWithImage(image)
    @image_view.contentMode = UIViewContentModeScaleAspectFit;

    @image_view.frame = [[50, 200], [200, 180]]
    view.addSubview(@image_view)
  end

  def start_camera
    if camera_present?
      @image_picker.sourceType = UIImagePickerControllerSourceTypeCamera
      presentModalViewControllerAnimated(@image_picker, animated:true)
    else
      show_alert("No camera available")
    end
  end

  def open_gallery
    @image_picker.sourceType = UIImagePickerControllerSourceTypePhotoLibrary
    presentModalViewControllerAnimated(@image_picker, animated:true)
  end

  def show_alert(message)
    alert = UIAlertView.new
    alert.message = message
    alert.addButtonWithTitle("OK")
    alert.show
  end

  # Check if the Camera is available or not
  def camera_present?
    UIImagePickerController.isSourceTypeAvailable(UIImagePickerControllerSourceTypeCamera)
  end
end
```

Attachment 21: 06_remote_api/Gemfile

```
source 'https://rubygems.org'

gem 'rake'
# Add your dependencies here:
gem "bubble-wrap"
```

Attachment 22: 06_remote_api/app/controllers/color_controller.rb

```
class ColorController < UIViewController
  attr_accessor :color

  def initWithColor(color)
    initWithNibName(nil, bundle:nil)
    self.color = color
    self
  end

  def viewDidLoad
    super

    self.title = "###{self.color.hex}"

    padding = 10
    offset = 64

    @info_container = UIView.alloc.initWithFrame([[0, offset], [self.view.frame.size.width,
60]])
    @info_container.backgroundColor = UIColor.lightGrayColor
    self.view.addSubview(@info_container)

    box_size = @info_container.frame.size.height - (padding * 2)

    # A visual preview of the actual color
    @color_view = UIView.alloc.initWithFrame([[padding, offset + padding], [box_size,
box_size]])
    @color_view.backgroundColor = String.new(self.color.hex).to_color
    self.view.addSubview(@color_view)

    text_field_origin = [
      @color_view.frame.origin.x + @color_view.frame.size.width + padding,
      @color_view.frame.origin.y
    ]
    @text_field = UITextField.alloc.initWithFrame(CGRectZero)
    @text_field.placeholder = "tag"
    @text_field.autocapitalizationType = UITextAutocapitalizationTypeNone
    @text_field.borderStyle = UITextBorderStyleRoundedRect
    @text_field.contentVerticalAlignment = UIControlContentVerticalAlignmentCenter
    self.view.addSubview(@text_field)

    @add = UIButton.buttonWithType(UIButtonTypeRoundedRect)
    @add.setTitle("Add", forState:UIControlStateNormal)
    @add.setTitle("Adding...", forState:UIControlStateDisabled)
    @add.setTitleColor(UIColor.lightGrayColor, forState:UIControlStateDisabled)
    @add.sizeToFit
    @add.frame = [
      [
        self.view.frame.size.width - @add.frame.size.width - padding,
        @color_view.frame.origin.y
      ],
      [
        @add.frame.size.width,
        @color_view.frame.size.height
      ]
    ]
  end
end
```

```

self.view.addSubview(@add)

add_button_offset = @add.frame.size.width + (2 * padding)
@text_field.frame = [
    text_field_origin,
    [
        self.view.frame.size.width - text_field_origin[0] - add_button_offset,
        @color_view.frame.size.height
    ]
]

@add.when(UIControlEventTouchUpInside) do
    @add.enabled = false
    @text_field.enabled = false
    self.color.add_tag(@text_field.text) do |tag|
        if tag
            refresh
        else
            @add.enabled = true
            @text_field.enabled = true
            @text_field.text = "Failed :("
        end
    end
end

table_height = self.view.bounds.size.height - @info_container.frame.size.height - offset
table_frame = [
    [0, @info_container.frame.size.height + offset],
    [self.view.bounds.size.width, table_height]
]
@table_view = UITableView.alloc.initWithFrame(table_frame, style: UITableViewStylePlain)
@table_view.autoresizingMask = UIViewAutoresizingFlexibleHeight
self.view.addSubview(@table_view)
@table_view.dataSource = self
end

def refresh
    Color.find(self.color.hex) do |color|
        self.color = color
        @table_view.reloadData
        @add.enabled = true
        @text_field.enabled = true
        @text_field.text = ""
    end
end

def tableView(tableView, numberOfRowsInSection:section)
    self.color.tags.count
end

def tableView(tableView, cellForRowAtIndexPath:indexPath)
    @reuseIdentifier ||= "CELL_IDENTIFIER"
    cell = tableView.dequeueReusableCellWithIdentifier(@reuseIdentifier)
    cell ||= UITableViewCell.alloc.initWithStyle(UITableViewCellStyleDefault,
reuseIdentifier:@reuseIdentifier)
    cell.textLabel.text = self.color.tags[indexPath.row].name
    cell
end
end

```

Attachment 23: 06_remote_api/app/controllers/search_controller.rb

```
class SearchController < UIViewController
  def viewDidLoad
    super
    self.title = "Search"
    self.view.backgroundColor = UIColor.whiteColor

    @text_field = UITextField.alloc.initWithFrame [[0,0], [160, 26]]
    @text_field.placeholder = "#abcabc"
    @text_field.textAlignment = UITextAlignmentCenter
    @text_field.autocapitalizationType = UITextAutocapitalizationTypeNone
    @text_field.borderStyle = UITextBorderStyleRoundedRect
    @text_field.center = [
      self.view.frame.size.width / 2,
      self.view.frame.size.height / 2 - 100
    ]
    self.view.addSubview(@text_field)

    @search = UIButton.buttonWithType(UIButtonTypeRoundedRect)
    @search.setTitle("Search", forState: UIControlStateNormal)
    @search.setTitle("Loading", forState: UIControlStateDisabled)
    @search.frame = [[0,0], [160, 26]]
    @search.center = [
      self.view.frame.size.width / 2,
      @text_field.center.y + 40
    ]
    self.view.addSubview(@search)

    @search.when(UIControlEventsTouchUpInside) do
      @search.enabled = false
      @text_field.enabled = false
      hex = @text_field.text
      # chop off any leading #s
      hex = hex[1..-1] if hex[0] == "#"

      Color.find(hex) do |color|
        if color.nil?
          @search.setTitle("None :", forState: UIControlStateNormal)
        else
          @search.setTitle("Search", forState: UIControlStateNormal)
          self.open_color(color)
        end
        @search.enabled = true
        @text_field.enabled = true
      end
    end

  def open_color(color)
    controller = ColorController.alloc.initWithColor(color)
    self.navigationController.pushViewController(controller, animated: true)
  end
end
```

Attachment 24: 06_remote_api/app/models/color.rb

```
class Color
  PROPERTIES = [:timestamp, :hex, :id, :tags]
  PROPERTIES.each do |prop|
    attr_accessor prop
  end

  def initialize(hash = {})
    hash.each do |key, value|
      if PROPERTIES.member? key.to_sym
        self.send((key.to_s + "=").to_s, value)
      end
    end
  end

  def self.find(hex, &block)
    BubbleWrap::HTTP.get("http://www.colr.org/json/color/#{hex}") do |response|
      result_data = BubbleWrap::JSON.parse(response.body.to_str)
      color_data = result_data["colors"][0]

      # Colr will return a color with id == -1 if no color was found
      color = Color.new(color_data)
      if color.id.to_i == -1
        block.call(nil)
      else
        block.call(color)
      end
    end
  end

  def tags
    @tags ||= []
  end

  def tags=(tags)
    if tags.first.is_a? Hash
      tags = tags.collect { |tag| Tag.new(tag) }
    end

    tags.each do |tag|
      if not tag.is_a? Tag
        raise "Wrong class for attempted tag #{tag.inspect}"
      end
    end

    @tags = tags
  end

  def add_tag(tag, &block)
    BubbleWrap::HTTP.post("http://www.colr.org/js/color/#{self.hex}/addtag/",
      payload:{tags: tag}) do |response|
      if response.ok?
        block.call(tag)
      else
        block.call(nil)
      end
    end
  end
end
```


Attachment 25: 06_remote_api/app/models/tag.rb

```
class Tag
  PROPERTIES = [:timestamp, :id, :name]
  PROPERTIES.each do |prop|
    attr_accessor prop
  end

  def initialize(hash = {})
    hash.each do |key, value|
      if PROPERTIES.member? key.to_sym
        self.send((key.to_s + "=").to_s, value)
      end
    end
  end
end
```

Attachment 26: 07_cocoa_pods/Rakefile

```
# -*- coding: utf-8 -*-
$.unshift("/Library/RubyMotion/lib")
require 'motion/project/template/ios'

begin
  require 'bundler'
  Bundler.require
rescue LoadError
end

Motion::Project::App.setup do |app|
  # Use `rake config` to see complete project settings.
  app.name = '07_cocoa_pods'
  app.interface_orientations = [:portrait]

  app.pods do
    pod "OpenWeatherMapAPI", "~> 0.0.5"
    pod "UIImage+PDF"
  end
end
```

Attachment 20: 07_cocoa_pods/controllers/weather_controller.rb

```
class WeatherController < UIViewController
  def loadView
    views = NSBundle.mainBundle.loadNibNamed "weather", owner: self, options: nil
    self.view = views[0]
    @view_handle = self.view
  end

  def viewDidLoad
    super

    self.title = "Weather"

    title_bar = view.viewWithTag 1
    # puts title_bar.items[0].rightBarButtonItem.inspect
    refresh_button = title_bar.items[0].rightBarButtonItem
    refresh_button.action = :refresh_weather

    @current_temp_label = view.viewWithTag 3
    @max_temp_label = view.viewWithTag 4
    @min_temp_label = view.viewWithTag 5
    @humidity_temp_label = view.viewWithTag 6
    @conditions_temp_label = view.viewWithTag 7
    @weather_icon = view.viewWithTag 8

    refresh_weather
  end

  def refresh_weather
    reset_values
    @weather_api = OWMWeatherAPI.alloc.initWithAPIKey("23138a16039ffef5a231ab39d10307a5")
    @weather_api.currentWeatherByCityName("Helsinki", withCallback: proc {|error, result|
      puts result.inspect
      @current_temp_label.text = format_temp(result["main"]["temp"])
      @max_temp_label.text = format_temp(result["main"]["temp_max"])
      @min_temp_label.text = format_temp(result["main"]["temp_min"])
      @humidity_temp_label.text = "#{result["main"]["humidity"]}%"

      if result["weather"] && result["weather"].any?
        set_weather_icon(result["weather"].first["icon"])
        @conditions_temp_label.text = result["weather"].first["description"]
      end
    })
  end

  def format_temp(temp)
    "#{temp.round(1)}°C"
  end

  def set_weather_icon(code)
    image = if %w(01d 01n 02d 02n 03d 03n 04d 04n 09d 09n 10d 10n 11d 11n 13d 13n 50d 50n).include? code
      UIImage.imageWithPDFNamed "climacons/#{code}.pdf", fitSize: @weather_icon.size
    else
      UIImage.imageWithPDFNamed "climacons/unknown.pdf", fitSize: @weather_icon.size
    end
    @weather_icon.setImage(image)
  end

  def reset_values
    [
      @current_temp_label,
      @max_temp_label,
      @min_temp_label,
      @humidity_temp_label,
      @conditions_temp_label
    ].each do |label|
      label.text = "..."
    end
  end
end
```