RubyMotion coding style

Table of Contents

Preface

This style guide is different from existing style guides ruby developers are used to. The main goal of this style guide is helping Objective-C developers start using RubyMotion. Our goal is getting rid of following two contradicting coding styles. We strongly encourage you reading the Objective-C coding style to gain a better understanding of this document.

Repository structure

The branch must have the structure described below regardless of the VCS used

- **app** – a directory for applications – both ***Objective-C*** and ***RubyMotion*** ones. Main application should contain no busyness logic. Please use static libraries and gems as much as possible.
- **lib** – a directory for ***Objective-C*** libraries containing busyness logic
- **lib-third-party** – a directory for third-party ***Objective-C*** libraries containing utility functions. Under ***git*** these must be included as ***submodules***
- **lib-ruby** – a directory for ***RubyMotion*** libraries containing busyness logic
- **lib-ruby-third-party** – a directory for third-party ***RubyMotion*** libraries containing utility functions. Under ***git*** these must be included as ***submodules***

Exception :

The main application must contain only xCode sub-projects that are responsible for GUI construction. Such sub-projects must be considered as wrappers for xCode's Interface Builder rather than stand alone projects.

0. Please use CamelCase for all declarations

Classes start with a ***capital*** letter.
Variables and methods start with a l***owercase*** letter
Constants are the only exception. They must be in the ***SCREAMING_SNAKE_CASE***

1. Spaces vs. Tabs

Use only spaces, and indent 2 spaces at a time. We use 2 spaces since they look best beside the“def” keyword.

2. Objective-C Method Overload Declarations

Please use the same rules as for Objective-C code. Method name parts must be aligned according to the colon sign. The same rule applies to invocations.

```
def initWithFrame(frame,
                  style: style )
  #implementation here
end
```

3. Ruby Method Declarations

Use `def` with parentheses when there are arguments. Omit the parentheses when the method doesn't accept any arguments.

4. Method Invocations

Please use parentheses explicitly when calling methods.

```
@table.reloadData  ## works but does not look like a function call
@table.reloadData() ## This is a lot better
```

Convenience constructors without parameters should be used without parentheses since they are rather constants than methods

```
UIColor.clearColor()  ## Not good. It's actually a constant
UIColor.clearColor  ## That's it!
```

When calling C functions, parentheses must always be used according to the RubyMotion's documentation http://www.rubymotion.com/developer-center/guides/runtime/#_functions

“Most functions in the iOS SDK start by a capital letter. For those who accept no argument, it is important to explicitly use parentheses when calling them, in order to avoid the expression to be evaluated as a constant lookup.”

5. Spacing and indentation

Spacing and indentation must be as close to Objective-C style as possible.

1. Please use spaces within parentheses

```
myObject.foo( arg1, arg2, arg3 )
```

2. Please separate brackets with spaces

```
myArray = [ 1, 2, 3, 4, 5 ]
```

3. Please ident “case” branches

```
      case  @dataType
        when SRAll
          imagePath = 'homePlaceholder.png'
        when SRNews
          imagePath = 'newsPlaceholder.jpg'
        when SRBlogs
          imagePath = 'blogPlaceholder.png'
      end
```

4. Please indent lambda implementation. Please prefer using multiline form.

```
queue.async do
 @loader.LoadFollowingItems(item.id, count:$itemsPageSize)
end
```

5. Language operators must be separated with spaces

6. Conditionals

- Avoid the ternary operator (?:)
- Prefer “Yoda style” boolean conditions

```
if ( 1 == n )
```

- Use “`unless`” keyword instead of negation operation within “if” statements
- Avoid reverse form of “if” operator

```
myObject.foo() if ( myStatement )
```

- Never use `unless` with `else`. Rewrite these with the positive case first.

```
# bad
unless success?
  puts 'failure'
else
  puts 'success'
end

# good
if success?
  puts 'success'
else
  puts 'failure'
end
```

7. Classes

Please use Ruby **modules** instead of Objective-C **protocols** and **enums**.

```
module SRDataType  ## this is enum
  SRAll   = 0
  SRNews  = 1
  SRBlogs = 2
end
```

Make use of all incapsulation features of ruby (`public`, `protected`, and `private`).

Native **ruby** methods must be as "***private***" as possible.
Objective-C overloads must be ***public*** to avoid interopability issues.

Please implement delegate callback handlers in different files for one class

```
class ReadingListController < UIViewController
  def viewDidLoad
    super
    # view did load logic
  end

  def shouldAutorotateToInterfaceOrientation(interfaceOrientation)
    return true
  end

  def shouldAutorotate
    return true
    end
end

class ReadingListController # this should be in another file
  # table view delegate + dataSource
  def numberOfSectionsInTableView(tableView)
    return 1
  end

  def tableView( tableView,
      numberOfRowsInSection: section)

    if ( @data )
      return @data.count
    end
    return 0
  end

# other code
end
```

8. Blocks and lambdas

Native Objective-C API can accept ruby lambdas as is.

```
queue.async do
 @loader.LoadFollowingItems(item.id, count:$itemsPageSize)
end
```

In order to call Objective-C blocks from the ruby code you should use a native extension :
https://github.com/dodikk/MotionBlocks

```
helloWorldMaker = NativeFactory.alloc.init.getHelloWorldProducer()
greetingString = helloWorldMaker.objc_BlockSend( [ 'Hello', 'Luke' ] )
puts( greetingString )  ## Hello world from Luke.
```

9. Memory Management

Ruby assignment operator retains objects. Object cycles are not supported by RubyMotion's garbage collector and cause memory leaks.
RubyMotion does not support weak references out of box. That's why some third-party Objective-C weak reference class must be used. You can pick any from the list below :

- https://github.com/farcaller/motion-memorymanagement
- https://github.com/dodikk/MotionBlocks

Please use **WeakRef** class as a replacement for Objective-C ***__weak*** keyword

```
 def setDelegate(delegate_)
   @delegate = RMWeakRef.refWithTarget( delegate_ )
 end

def getDelegate
 return @delegate.target
end
```

Use ruby autorelease pools for memory critical sections

```
Kernel.autorelease_pool do
 ## memory critical code
end

# non memory critical code
```

10. One more thing

Always use “return” statement if a method is supposed to return some value

Prefer double-quoted strings. Interpolation and escaped characters will always work without a delimiter change, and `'` is a lot more common than `"` in string literals.

Never use `for`, unless you know exactly why. Most of the time iterators should be used instead. `for` is implemented in terms of `each` (so you're adding a level of indirection), but with a twist - `for` doesn't introduce a new scope (unlike `each`) and variables defined in its block will be visible outside it.

```
arr = [1, 2, 3]

# bad
for elem in arr do
  puts elem
end

# good
arr.each { |elem| puts elem }
```