OAB- Coding convention Manual for iOS developers.

PREPARED BY:

Gaurav Taywade

Chief Software Architect (CSA)– OAB STUDIOS

Table of contents :

Revision History

Date	Version	Revision description	Author
13 August 2014	1.0	Initial Draft	Gaurav Taywade
	1.0.	Review	Sukesh Jadhav
	1.0	Review	Sudhir Chavan
	1.1	Final draft	Gaurav Taywade

Objective and Scope

The objective of this document is to describe the suggested coding conventions to be followed for OAB iOS Development team using Objective-C. This document must be part of iOS application architecture.

The document will describe:

- Inline code comments
- Spacing and Indentations
- Naming conventions

Code Comments

2.1 File Header Comments

XCode IDE automatically generates file header comments on adding new file. The comments block should contain file name, project name, file description, creation date, first author name, and company information. See the sample snippet for header comments, you can change company name from XCode settings.:

```
//  File name.ext
//  Project name
//
//  Created by <author name> on m/d/y.
//  Copyright 20xx __CompanyName__. All rights reserved.
//
```

Code Comments

2.2 Classes, Categories and Protocols

- **Class/Interface definition comments**

```
/**
   @class ClassName
   @inherits Base class
   @conforms Comma separated list of protocol name those this class conforms to
   @description description of the class - may be multiline description
 */
```

- **Category definition comments**

```
/**
   @category Category name
   @class class Name of this category
   @description description of the category
 */
```

- **Protocol definition comments**

```
/**
   @protocol ProtocolName
   @description details and usage of this protocol
 */
```

Code Comments

2.3 Methods, Properties and Instance variable

- **Method definition comments**

```
/**
   @method methodName
   @description method functionality description
   @param param1 name and description
   @param param2 name and description
   @returns description of returning object
 */
```

- **Property definition comments**

```
/**
   @property property name
   @description property description
 */
```

- **Instance Variable comments**

```
 /**
    @var instance variable name
    @description ivar description
  */
```

Code Comments

2.4 Enumerations and constants

- **Enumeration definition comments**

```
/**
   @enum Enumeration name
   @description general description of enum
 */
```

Each constant in enum should have a single line comment on right side of its definition e.g.

```
typedef enum {
        OBEnumeratedTypeFirst    // description of OBEnumeratedTypeFirst
        OBEnumeratedTypeSecond   // description of OBEnumeratedTypeSecond
} OBEnumeratedType;
```

- **Constant definition comments** - constants are global variables with const (or #define) qualifier

```
/**
   @constant variable Name and description
 */
```

Spacing and Indentations

Spacing and indentations make the code easy to read and thus easy to understand. We will use JAVA style spacing and indentations. Following are some basic rules:

- Write each statement on new line.
- Declare each variable on new line
- Synthesize each property on new line.
- There must be space after each comma (,)
- There should be space after semicolon (;) only if it does not indicate an end of statement.
- Write space between #import (or #include) and header file name.

Spacing and Indentations Cont…

- Each code block should be indented 4 spaces ahead of its parent code block, like:

```
- (void)methodname {
   // TODO: write an inner block
   if (some condition) {
      // write down the code
}
}
```

- Pointer variable accessor methods should be declared like:

```
- (TypeOfVeriable *)ptrVariableName; // instance variable getter
+ (TypeOfVeriable *)ptrVariableName; // static/class variable
```

- Notice the spacing between minus/plus sign and starting parenthesis '(', and notice the spacing between TypeOfVeriable and asterisk '*'. Also note the space between ending parenthesis ')' and variable name.
- The same convention should be followed by properties and methods declarations and definitions. Like:

```
@property (nonatomic, assign) (TypeOfProperty *)propertyName;

- (ReturnType *) assentialMethodSignature:(FirstParamType *)firstParameter
signatureForSecondParam:(SecondParamType)secondParam; // Instance method declaration

+ (ReturnType *) assentialMethodSignature:(FirstParamType *)firstParameter
signatureForSecondParam:(SecondParamType)secondParam; // static/class method declaration
```

**** Please notice carefully all the spaces in the above examples.**

Spacing and Indentations Cont…

- While calling methods the very long signature, you may use multiple lines but then you should write each parameter on a new line, like:

```
[instance veryLongMethodSignature:firstParameterValue
                secondParamName:secondParamValue
                 thirdParamName:thirdParamValue];
```

Please notice the colons before each parameter value, which aligned vertically. This the default feature of XCode IDE

- Switch case structure should be written in the following format:

```
switch (expression to be tested) {
    // write comments for first case OR a blank line
    case testValue1:
        // this line is indented ahead of 'case' line
        break;
    // write comments for next case or leave it as blank line
    default:
        // write default case code
        break;
}
```

Spacing and Indentations Cont…

- Class member or local pointer variable names should have no space between asterisk '*' and variable name, like:

```
TypeOfVariable *variableName;
```

- There should be single space between class name and start parenthesis of category name.

- All conditional and iterative code blocks should be written inside braces {}.

- Indent size should be set to 4 spaces

- Starting brace '{' should be placed at the end of the line of code which require braces, like:

```
if (some condition) {
    // some comments
}
```

- Line next to the starting brace should be left blank or should be used for writing comments

Naming Conventions

Good naming convention makes the code easy to write, understand and it reduces the overhead of reading reference documents for all of the API stuff. Things that should be considered while naming the Objective-C constructs:

- Names should be clear and brief.
- Don’t use abbreviations for names except some very common abbreviations like: rect, info etc.
- Try to avoid ambiguous naming. The name should reflect one and only one meaning.
- Make the names consistent for objects, methods, properties those have the same roles and/or behaviors in various classes.
- Don’t use underscores for as word separator in multiple word names.
- If a name contains multiple words, then each new word will be started with a Capital letter.
- Names for Classes, Protocols, Enumerations, C Functions should have a literal prefix like Cocoa class NS, UI, MF etc.

Naming Conventions

4.1 - Prefixes for Type Names

- All classes, protocol, Enumerations, C Functions should have a 2,3 letter prefix representing either the Project or OAB Studios proprietary.
- We will use Project short code for project specific type names, e.g. DC for Dormchat, TH for Thirst, and BS for Bodystream project.
- All type names of reusable stuff will be prefixed by OA (OAB)
- All custom controls that we build inside OAB will need to have letter prefix OA, e.g. OACustomSlider, OACameraControl.

Naming Conventions

4.2 Naming Classes, Categories and Protocols

- Class names should be a noun clearly indicating the type and behavior of the class or instance. The noun should start with a Capital letter. Examples:

  DCAssetPickerController, THSlideViewController

- Naming convention for Protocols is same as of classes, but the protocol name may specify some verb too. Naming a protocol should define its usage behavior too. Examples:

  DCBookShelfDataSource, OASlideViewDelegate, OACommunicationManager

- Category name should follow the syntax below:

  [ClassName] + [CategoryDescription] + Additions

- And the category header and implementation file names should look like:

  [ClassName]+ [CategoryDescription].[ext]

Naming Conventions

4.3 Naming Enumerations and Constant

- The enumeration type should clearly specify where it should be used. Examples:

```
typedef enum {
    DCControllerStateNone,    // description of None state
    DCControllerStateEditing, // description of Editing state
    DCControllerStateDisabled    // description of Disabled state
} DCControllerState;
```

- Please notice the name of each constant, inside DCControllerState enumeration type definition, starts with the enumeration type name.

- Constants defined by #define should be named in all capital letters and underscores should used to as word separator if the name contains multiple words. E.g.

```
#define    DONT_USE_HASH_DEFINE    0
```

- Generally, #define constants are discouraged, rather we should use constants defined by enumerations or qualified by const keyword. Constants defined using const keyword should be prefixed with small letter 'k' and should follow enumerations like naming convention for the rest of the name. E.g.

```
const CGFloat kDCAnimationSpeed = 1.6f;
```

Naming Conventions

4.4 Naming Methods

Generally Objective-C methods follow camel case naming convention. In Objective-C, methods are invoked through messages and named like a sentence OR phrase in any spoken language. I will strongly recommend reading Apple's Coding Guidelines for Cocoa for proper understanding of naming. Here are few rules for naming methods:

- Use meaningful names describing the behavior and resultants of the method.
- Methods representing actions should start with a verb.
- Don't use 'get' for getters but setter methods should start with 'set'. Use 'is' or 'can' or 'should' for getters returning Boolean values. You may still use 'get' before a method name if the method returns object by some manipulation.
- If a property name is 'namingProperty' of type 'ExampleType' then

Naming Conventions

4.4 Naming Methods - Cont..

- Its setter signature will be:

```
- (void)setNamingProperty:(ExampleType)aNamingProperty;
```

and its getter will be:

```
- (ExmapleType)namingProperty;
```

- If a Boolean property of some instance defines some eligibility of the instance then its getter method may look like:

```
- (BOOL)isPropertyName;
```

- And if a Boolean property of some instance defines a capability of the instance then its getter method may look like:

```
- (BOOL)canPropertyName; // OR
- (BOOL)sholdPropertyName;
```

- User keywords (some word before the colon describing the argument) before all arguments.

Naming Conventions

4.4 Naming Methods - Cont..

- Make the word before the argument describe the argument. E.g.

```
- (id)viewWithTag:(int)aTag; // is the Right convention
- (id)taggedView:(int)aTag; // is convention
```

- Argument names follow the same naming convention as methods. Don't use 1 or 2 letters or an abbreviation as an argument name.
- Avoid usage of helping verbs in method naming.
- Avoid usage of only prepositions (and/or) to connect method name keywords.
- For delegates, the first part of the method name should be the sender of the event/ notification followed by some identification of what notification is being sent. Use 'will', 'did', and 'should' to describe the sequence of event.

Naming Conventions

4.5 Naming Functions

- C function names start with a prefix, appended with the specific Module or structure name. The prefix rule is same as for classes. Function names start with capital letters and for the remaining part of name (following Prefix and Object name) follow the same naming conventions as Objective-C methods. The naming template should look like:

```
PREFIX + ModuleOrObjectName + ProperFunctionName
```

e.g.

```
CGGraphicsContextDrawBitmap(…);
```

Where

CG is the PREFIX

GraphicsContext defines the object for which the method is being defined.

DrawBitmap is the unique part of function name

Naming Conventions

4.5 Naming Exceptions

Use the following simple template to name exceptions:

[PREFIX] + [UniqueName] + Exception

e.g

DCIndexOutOfBoundException

Naming Conventions

4.5 Naming Variable and Properties

- Generally, variable and properties names should start with a small letter and use mixed case to delimit words. Here are some rules:
- Use underscore as prefix in instance variable names.
- Don’t use underscores in start of local variables, and property names.
- Don’t use Hungarian notation to describe variable type and scope etc. Rather use descriptive names to make the name understandable. Don't worry about saving horizontal space, as it is far more important to make your code immediately understandable by a new reader
- Make sure the name of the instance variable or property concisely describes the attribute stored
- If you have defined an instance variable, don’t use the same name for some method’s argument name because it hides the class member access in that method. In such cases use ‘a’ before the argument name.

Reference

- Apples Coding Guidelines for Cocoa
  http://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/CodingGuidelines/CodingGuidelines.html#//apple_ref/doc/uid/10000146-SW1

- Folio3 objectiveC coding conventions document by Mr. Asif Kamboh

- Java coding conventions by Oracle
  - http://www.oracle.com/technetwork/java/codeconvtoc-136057.html